MASTER IN COMPUTER SCIENCE

MPTCP On Rails

Miny, Laurent

Award date:
2017

Awarding institution:
University of Namur

Université de Namur
Faculté d'informatique
Année académique 2016–2017

MPTCP On Rails

Laurent MINY

Promoteur : _____ (Signature pour approbation du dépôt - REE art. 40)
Laurent SCHUMACHER

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Résumé

L'idée de ce mémoire est d'effectuer des mesures sur l'impact du Multipath TCP (MPTCP) lors de *streaming* audio dans les transports sur voie ferrée (trains, trams et métros). Pour les réaliser, nous avons élaboré un dispositif expérimental qui comporte deux interfaces. L'une est reliée au réseau cellulaire de manière continue, alors que l'autre profite par intermittence des hotspots Wi-Fi publics mis à disposition lors des trajets. Ces tests ont permis de mettre en évidence certains problèmes liés à l'utilisation du gestionnaire de réseau Debian qui ne réagit pas suffisamment rapidement au changement de borne Wi-Fi. Ce problème, lié à une périodicité de *scan* des réseaux insuffisante, entrave le fonctionnement des scénarios mobiles. L'authentification aux portails d'accès des hotspots pose également problème lors de l'utilisation de MPTCP. Il est alors nécessaire de préciser l'IP source lors de la récupération de la page d'authentification afin d'envoyer le paquet sur l'interface Wi-Fi et non sur l'interface cellulaire déjà reliée à Internet. Enfin, l'utilisation du *tethering* avec un smartphone, masque le *handover* Wi-Fi au niveau du dispositif et il convient notamment de s'assurer que la table NAT est bien "flushée" lors du changement d'adresse IP.

Abstract

The goal of this master thesis is to measure the impact of Multipath TCP (MPTCP) during audio streaming on rails (trains, trams and metros). To do that, we put in place an experimental set-up with two interfaces. The first one is linked on the cellular network and the second one is connected to public Wi-Fi hotspots available during the trips. These tests helped to discover some problems linked to the network manager of Debian that has not be designed for mobility purposes. It hardly detects the switch of access point. This problem is due to an insufficient network scan periodicity. The authentication page of portals was also laborious. One solution is to specify the IP source to be able to get the authentication page of the portal. This trick forces the packet not going to the cellular interface already connected to the Internet. Finally, the use of tethering with a smartphone mask the Wi-Fi handover from the experimental device point of view. It is, for example, useful to check if the NAT table has been correctly flushed when switching the networks.

Remerciements

Ce mémoire est l'aboutissement de nombreuses expérimentations, investigations et améliorations afin d'obtenir des résultats probants.

J'aimerais tout d'abord remercier Monsieur Schumacher, mon promoteur, pour le soutien continu tout au long de la rédaction de ce mémoire et pour la confiance sans faille qu'il m'a accordée.

Merci également à l'équipe de développement du kernel Linux MPTCP de l'UCL, pour les nombreuses réponses obtenues sur la *mailing list*.

Je remercie de même toutes les personnes que j'ai contactées et qui m'ont permis de progresser dans ce mémoire.

Mes remerciements vont tout particulièrement à Valentin Desplanque et Martin Houry pour leurs remarques techniques et bons conseils.

Je tiens à remercier aussi mon entourage et mes amis de l'UNamur pour la relecture et les encouragements si précieux.

Merci également à vous, cher lecteur, pour l'attention que vous porterez à ce mémoire.

Table des matières

Table des figures

Liste des Abréviations

ACK	**Ack**nowledgement
AP	**A**ccess **P**oint
DASH	**D**ynamic **A**daptive **S**treaming over **H**TTP
DHCP	**D**ynamic **H**ost **C**onfiguration **P**rotocol
FIN	**Fin**alize
HMAC	**H**ash-based **M**essage **A**uthentication **C**ode
IBPT	**I**nstitut **B**elge des services **P**ostaux et des **T**élécommunications
IP	**I**nternet **P**rotocol
IPS	**I**ntrusion **P**revention **S**ystems
MAC	**M**edia **A**ccess **C**ontrol
MPTCP	**M**ulti**P**ath **T**ransmission **C**ontrol **P**rotocol
NAT	**N**etwork **A**ddress **T**ranslation
RST	**Res**e**t**
RTO	**R**etransmission **T**ime**O**ut
RTP	**R**eal **T**ime Trans**p**ort
RTT	**R**ound-**T**rip **T**ime
SNCB	**S**ociété **N**ationale des **C**hemins de fer **B**elges
SSID	**S**ervice **S**et **Id**entifier
STA	**Sta**tion
STIB	**S**ociété des **T**ransports **I**ntercommunaux de **B**ruxelles
SYN	**Syn**chronize
TCP	**T**ransmission **C**ontrol **P**rotocol
TGV	**T**rain à **g**rande **v**itesse

Chapitre 1

Introduction

L'objectif de ce mémoire est d'entreprendre des mesures dans des scénarios de transports sur voie ferrée afin de déterminer s'il est possible de tirer parti du Multipath TCP (MPTCP) et d'en observer les impacts.

Pour réaliser ces mesures, l'utilisation d'une interface 4G combinée à une interface Wi-Fi se connectant par intermittence à des hotspots a été envisagée.

Dans ce mémoire, nous décrivons dans un premier temps le MPTCP et dressons un état de l'art de cette technologie au chapitre 2.

Le chapitre 3 décrit le dispositif expérimental. Les choix de l'architecture y sont exposés et un aperçu des recherches menées dans des contextes proches de celui de ce mémoire est présenté.

Pour mettre en place les mesures, il a été nécessaire de faire un tour d'horizon de l'environnement, en terme de 4G et de Wi-Fi, disponible dans les différents transports sur voie ferrée que sont les trains, trams et métros. Le chapitre 4 tente d'expliquer ces mesures tout en montrant la difficulté des conditions expérimentales.

Le chapitre 5 présente les scénarios de *streaming* audio envisagés dans les différents environnements. Les résultats de ces mesures sont ensuite analysés.

Enfin, pour mettre en balance les résultats obtenus lors des précédents chapitres, le chapitre 6 propose une réflexion sur le travail effectué. Plus précisément, une critique du dispositif expérimental est réalisée, et des parades possibles aux problèmes rencontrés sont présentées.

Chapitre 2

Multipath TCP

2.1 Contexte

Internet évolue continuellement et doit faire face à de nouveaux défis. La pile protocolaire TCP/IP conçue au début de l'Internet a de plus en plus de mal à subvenir aux besoins actuels. Alors qu'à l'époque de sa conception, les terminaux ne comportaient en général qu'une seule interface, [3] aujourd'hui il n'est pas rare de trouver des systèmes équipés de plusieurs interfaces. Effectivement, le nombre de smartphones qui disposent d'une connexion internet cellulaire et du Wi-Fi croît d'année en année. [31]

Les attentes des utilisateurs sont de pouvoir utiliser ce *multihoming* [1] pour combiner les connexions et ainsi améliorer la redondance et la performance du réseau. [3] Cependant, TCP est limité à une seule interface par connexion, ce qui implique que l'utilisation de plusieurs interfaces en TCP n'est pas "transparente" pour la couche application.

Le protocole MPTCP a été développé pour prendre en compte les limitations de TCP. L'intérêt principal est de pouvoir mettre en place un protocole qui s'intègre de manière optimale dans l'architecture actuelle d'Internet sans devoir modifier les périphériques existants éparpillés sur le réseau. [4]

2.2 Description

Multipath TCP (MPTCP) est un ensemble d'extensions apportées à TCP dans le but de permettre le transport d'informations sur plusieurs chemins TCP appelés également "*subflows*". Concrètement, MPTCP s'occupe de mettre en place les différents *subflows*, de les gérer, de rassembler les informations et de terminer les sessions.

L'utilisation de ces différents chemins a plusieurs avantages [19] :

1. Le *multihoming* permet de maintenir plusieurs connexions Internet dans le but d'améliorer la fiabilité du réseau.

— Améliorer le *throughput* : leur combinaison peut apporter un meilleur débit à la couche application ;
— Améliorer la résilience : une interface peut servir de *backup* en cas de panne sur un *subflow*. Les chemins sont donc interchangeables et les segments peuvent être ré-envoyés sur ceux disponibles.

Ce protocole est entièrement rétro-compatible avec le protocole TCP, couramment utilisé, afin de faciliter son déploiement. [12]

Cette technologie offre comme avantage l'amélioration de l'expérience utilisateur, par l'amélioration du débit grâce à la combinaison de plusieurs interfaces, ainsi que par la résilience face aux pannes.

Les objectifs du Multipath TCP comme définis dans [28] sont :

1. une application doit pouvoir fonctionner en TCP comme en MPTCP sans devoir être modifiée ;
2. MPTCP doit fonctionner dans tous les cas où TCP est d'application. S'il a un problème quelconque sur un des *subflows*, la connexion doit continuer à fonctionner aussi longtemps qu'il y aura une connexion sur une autre interface ;
3. MPTCP doit fonctionner au moins aussi bien que TCP, notamment au niveau du contrôle de congestion [38] ;
4. MPTCP doit pouvoir être implémenté dans un système d'exploitation sans utilisation de mémoire ou de *processing* excessif.

Nous allons maintenant expliquer de manière simplifiée l'architecture du Multipath TCP.

2.3 Architecture

MPTCP agit comme une couche intermédiaire entre l'interface du socket et une ou plusieurs connexions TCP comme montré dans la figure 2.1

Toutes les informations propres à MPTCP sont stockées dans les options de TCP et permettent [24] :
— d'établir une connexion MPTCP ;
— d'ajouter et de retirer des subflows à une connexion MPTCP ;
— de transmettre des données sur la connexion MPTCP.

Toutes les opérations relatives à la connexion MPTCP sont stockées dans une seule et unique option TCP avec un sous-type pour chaque message. Passons en revue le but de chacun de ces messages.

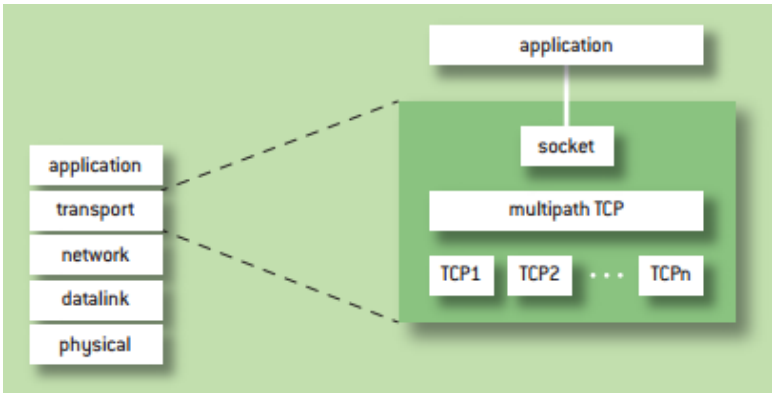

FIGURE 2.1 – Architecture de MPTCP (Source : [24])

2.3.1 Handshake initial

Le client crée un premier subflow TCP sur une interface à l'aide du *three-way handshake*. La figure 2.2 représente cet échange. Chaque paquet `SYN`, `SYN/ACK` et `ACK` contient l'option `MP_CAPABLE`. Cette option indique à l'hôte distant que l'expéditeur supporte le MPTCP.

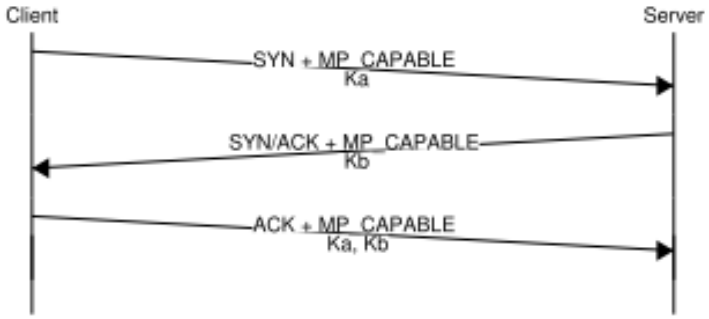

FIGURE 2.2 – Handshake du premier *subflow* (Source : [10])

Lors de ce *handshake*, un échange de clés de 64 bits générées de part et d'autre de la connexion permet d'authentifier l'ajout futur de nouveaux *subflows* à cette connexion. Cette clé n'est envoyée en clair qu'une seule fois, lors de l'initialisation de la connexion MPTCP. Ensuite un *token* sur 32 bits, généré grâce à un *hash* de la clé [12], est utilisé pour identifier la connexion.

Le troisième message du *three-way handshake* permet de confirmer l'utilisation de MPTCP avec l'option `MP_CAPABLE` contenant les deux clés. Cette dernière validation est utile dans le cas où une *middlebox* (e.g. NAT ou Firewall) supprimerait le `MP_CAPABLE` du segment `SYN/ACK`. Si c'est effectivement le cas et qu'un des paquets

ne présente pas l'option, MPTCP se rabat alors sur le TCP classique qui opère sur un seul chemin. [12]

2.3.2 Handshake des *subflows* additionnels

Une fois la première connexion réalisée avec l'option `MP_CAPABLE`, l'ajout de nouveaux *subflows* est possible. Pour ce faire, le client (ou le serveur [12]) peut démarrer un nouveau *subflow* sur une autre interface avec un *three-way handshake* TCP classique mais comportant l'option `MP_JOIN`, comme illustré sur la figure 2.3. Cette option permet d'identifier la connexion MPTCP grâce au *token*.

Ce recours à un *token* comme identifiant est motivé par la présence de *middleboxes*. L'utilisation des 5 tuples classiques [2] pour identifier la connexion aurait en effet posé problème avec les NATs.

Le premier SYN contient donc le *token*, qui a été généré avec la clé du serveur, de même qu'un nombre aléatoire (*nonce*) utilisé pour empêcher les attaques par rejeu.

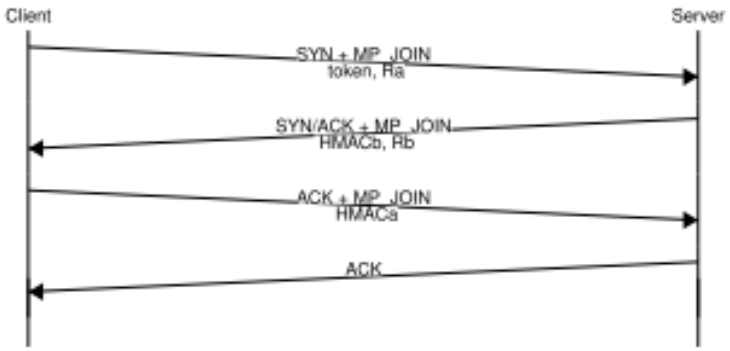

FIGURE 2.3 – Handshake des *subflows* additionnels (Source : [10])

Ensuite, si le *token* correspond à une connexion au niveau du serveur, celui-ci renvoie un `SYN/ACK` contenant également l'option `MP_JOIN`. Pour authentifier l'ajout du *subflow*, le serveur calcule un *Hash-based Message Authentication Code* (HMAC) `HMACb` sur base du *nonce* reçu `Ra` et des clés échangées lors du *handshake* initial `Ka` et `Kb`. Ce dernier est renvoyé au client avec un nonce `Rb`.

Le client vérifie ensuite la conformité du `HMACb` afin de s'assurer que le serveur possède les clés `Ka` et `Kb`. [23] Cette vérification prouve

2. Les 5 tuples sont : Le protocole, l'IP$_{src}$, l'IP$_{dest}$, le port$_{src}$ et le port$_{dest}$

que le serveur est bien celui qui a participé au *handshake* initial et permet de prévenir les attaques de spoofing à l'aveugle de paquets d'un adversaire qui veut détourner la connexion existante. [28] Le client acquitte ensuite le troisième paquet par un `ACK` avec l'option `MP_JOIN` incluant le nouveau `HMACa` calculé sur base du *nonce* `Rb` renvoyé par le serveur. Enfin, le serveur vérifie ce *hash* et renvoie un `ACK`.

2.3.3 Retirer un *subflow*

Pour pouvoir supprimer un *subflow*, il faut pouvoir l'identifier. Chaque adresse IP dispose d'une `Address ID` unique au sein de la connexion. Elle identifie une interface particulière de l'expéditeur et est utilisée pour retirer un *subflow* tout en fonctionnant à travers le NAT. Cet `Address ID` peut être envoyée par l'intermédiaire d'un segment `MP_JOIN` ou `ADD_ADDR`.

L'option TCP `ADD_ADDR` annonce les adresses supplémentaires sur lequel un hôte peut être atteint. Ce signal MPTCP peut être utilisé à n'importe quel moment de la connexion en fonction de l'expéditeur pour activer plusieurs chemins et/ou quand des chemins deviennent disponibles. [12]

Pour retirer un *subflow*, un signal similaire, `REMOVE_ADDR` est utilisé. Cette option est particulièrement utile pour avertir le receveur qu'une interface n'est plus disponible (e.g. lorsqu'un smartphone perd la connexion Wi-Fi en s'éloignant du point d'accès auquel il était connecté). [10] Lorsque le destinataire reçoit ce message, il vérifie que le chemin n'est plus utilisé en envoyant un `TCP keepalive` sur le chemin. S'il n'y a aucune réponse, il clôture tous les *subflows* TCP qui utilisent cette `Address ID` en envoyant des `RST`'s. [12]

2.3.4 Fermeture de connexion

En utilisant TCP, lorsqu'un `FIN` est envoyé sur le réseau, cela signifie que l'expéditeur n'a plus de données à envoyer. Avec MPTCP, un `FIN` a seulement un impact sur le *subflow* sur lequel il a été envoyé.

Pour MPTCP il y a également un mécanisme similaire appelé `DATA_FIN` comme illustré sur la figure 2.4. Ce message a la sémantique et le comportement d'un `FIN` TCP classique sauf qu'il s'applique au niveau de la connexion MPTCP. Les deux messages, `DATA_FIN` et `FIN`, ne sont pas liés. La seule utilisation simultanée possible de ceux-ci ne peut être réalisée que s'il n'y a plus aucune donnée restante sur les autres *subflows*.

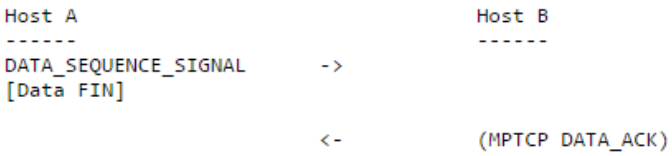

FIGURE 2.4 – Fermeture de connexion MPTCP (Source : [12])

Dès lors que le `DATA_FIN` a été acquitté, tous les *subflows* restants sont alors fermés avec des `FIN` classiques. La connexion est alors considérée fermée si les deux `DATA_FIN`'s, pour fermer la connexion dans les deux sens, ont été acquittés.

2.3.5 *Subflows* de backup

Un *subflow* de backup peut être utilisé typiquement dans le cadre d'une connexion via un smartphone, pour limiter les coûts liés à l'internet mobile lorsqu'une connexion Wi-Fi est disponible. Pour bénéficier de cette fonctionnalité, le *flag* `B` de l'option `MP_JOIN` lors du *handshake* doit être mis à 1. Lorsque cette option est validée, l'expéditeur demande à l'hôte distant de ne plus envoyer de données sur ce *subflow* s'il n'y a pas d'autre *subflows* disponibles (qui ne sont pas en mode "*backup*").

Le signalement d'un *subflow* de backup peut également être réalisé lorsqu'une connexion est déjà établie. La priorité d'un *subflow* peut être modifiée via l'option `MP_PRIO` en indiquant le *flag* `B` à 1. Une indication importante est que ce signal ne fonctionne que dans une seule direction. Par exemple, si le signal est envoyé du serveur vers le client, rien n'empêche le client d'envoyer des données sur le *subflow* moins prioritaire. [12]

2.3.6 Gestion des chemins

Un nœud mobile comme un smartphone ou un ordinateur dans les transports en commun disposant de plusieurs interfaces comme la 4G et le Wi-Fi n'utilise généralement qu'une seule interface à la fois. L'utilisateur aimerait tirer parti de ces différentes interfaces. Nous pouvons distinguer trois points de vues [26] :

1. le premier est de privilégier la performance du transfert de données afin d'obtenir le débit le plus rapide ;
2. le deuxième point de vue est de favoriser la durée de vie de la batterie. Il faut alors faire un compromis entre la performance et la durée de vie ;

3. enfin, le troisième facteur est le prix. Certains réseaux comme celui de la 4G fonctionnent avec une structure de coût qui tient compte du nombre de bits ou paquets téléchargés.

MPTCP prend ces plusieurs points de vues en considération et propose différents mécanismes de *handover* afin d'alterner entre les différents réseaux sans perte de connexion. Plusieurs *handovers* sont possibles [24, 26] :

— **mode *Full-MPTCP*** (*fullmesh*) : mode basé sur le principe *make-before-break* dans lequel tous les chemins sont utilisés. Ce mode est dédié aux utilisateurs qui veulent obtenir des gains de performances au niveau du débit. C'est le mode qui consomme le plus d'énergie mais cette consommation est à relativiser. Car tout d'abord, l'affichage graphique consomme plus que les interfaces radios. Ensuite, lorsque l'utilisateur attend que la page se charge, l'utilisation de deux interfaces réduit l'utilisation de l'écran et donc, dans une certaine mesure, réduit également la consommation d'énergie [24] ;

— **mode *Backup*** : Des connexions TCP sont ouvertes sur toutes les interfaces comme pour le mode précédent. La différence se situe au niveau du transfert des données. Celui-ci s'effectue uniquement sur les *subflows* qui ne sont pas marqués comme étant des *subflows* de backup [3]. Ce mode peut être utilisé pour réduire les coûts ainsi que la consommation d'énergie en utilisant les interfaces les moins chères. C'est également préférable de favoriser l'utilisation du Wi-Fi quand il est disponible et de n'utiliser la 4G que lorsque celui-ci n'est plus accessible [26] ;

— **mode *Single-Path*** : Ce mode se base sur l'approche *break-before-make* qui autorise un petit moment pendant lequel aucun *subflow* n'est actif. MPTCP transfère les données sur une seule interface. Quand celle-ci tombe en panne, une autre interface est alors activée et un *subflow* est créé sur celle-ci. Une fois le *subflow* créé, toutes les données perdues y sont alors retransmises. Le transfert au niveau de la couche application continue donc sans avoir été interrompu. L'inconvénient de ce mode par rapport au mode *Backup* est le temps perdu dans la mise en route d'un autre *subflow* au moment d'une panne.

2.4 Transport *multipath* fiable

Pour échanger des données sur plusieurs *subflows*, une première solution serait d'utiliser les numéros de séquences de TCP en conservant le même espace entre les deux *subflows*. Cette idée serait fonctionnelle dans un monde sans *middleboxes* (e.g. NAT, Firewall, IPS). Or nous ne pouvons pas ignorer celles-ci qui sont présentes

3. les *subflows* de backup sont expliqués en détail au point 2.3.5

dans l'Internet actuel. Le problème avec ces *middleboxes*, c'est que certaines peuvent analyser les paquets et détecter un écart dans les numéros de séquence d'un *subflow* et décider de faire un *drop* des paquets.

Pour surmonter cette difficulté, MPTCP propose deux niveaux de numéros de séquences. Le premier relatif à chaque *subflow* et le second au niveau du Multipath TCP. Ce dernier est appelé "Data sequence number". Les paquets reçus sur chaque *subflow* sont réordonnés à l'aide du numéro de séquence TCP et MPTCP s'occupe du ré-ordonnancement au second niveau grâce au "Data sequence number"

Si des données sont perdues sur un *subflow* TCP, elles sont renvoyées de manière classique avec un *fast retransmit* après un certain *timeout*. Dans le cas où un *subflow* tombe en panne, le client détecte la perte du *subflow*, soit grâce au RTO (Retransmission TimeOut) de TCP, soit parce que l'adresse IP a disparu ou que l'interface s'est éteinte. Dans ces deux cas de figures, les données sont retransmises sur un autre *subflow* choisi par le *scheduler*. Le *scheduler* par défaut envoie les paquets sur le *subflow* avec le RTT (Round-Trip Time) le plus bas et qui dispose encore d'espace dans sa fenêtre de congestion. La retransmission sur un autre *subflow* est appelée "réinjection". [1]

2.5 Proxies MPTCP

Étant donné que MPTCP n'est actuellement pas encore massivement déployé, il est nécessaire de mettre en place des *proxy* MPTCP pour pouvoir utiliser les services existants tout en profitant des avantages que procure le *multipath*. L'utilité est double : mettre en place progressivement le *multipath* sans devoir mettre à jour les systèmes déjà existants et tester son utilisation.

DE CONINCK et BAERTS ont d'ailleurs réalisé un mémoire [10] dans lequel ils examinent l'impact du MPTCP sur smartphone lors d'envoi de données avec les applications existantes. Ils ont été contraints de mettre en place un *proxy* MPTCP pour permettre à celles-ci de tirer profit des interfaces Wi-Fi et 4G des smartphones.

Le principe du *proxy* est de générer une connexion MPTCP entre l'hôte mobile et le *proxy*. Ce dernier se connecte alors au serveur en TCP et effectue en ce sens une "conversion" entre MPTCP et TCP.

Dans le contexte où le *proxy* pourrait être utilisé par les fournisseurs d'accès à Internet, comme proposé dans l'article [29], le *proxy* tenterait d'abord d'établir une connexion MPTCP vers l'hôte distant. Si celle-ci est établie, une connexion s'opère directement entre les 2

hôtes sans utiliser le *proxy* comme intermédiaire. Enfin, dans le cas où la connexion MPTCP n'aboutit pas, le *proxy* agit alors comme dans la solution proposée dans le paragraphe précédent.

Reprenons plus en détails ces deux modes de fonctionnement du *proxy* MPTCP [29] :

1. **hôte distant supporte MPTCP** : le *proxy* indique à l'hôte mobile l'adresse IP de l'hôte distant disponible pour créer un *subflow* MPTCP. Une connexion est établie avec l'hôte distant et la connexion avec le *proxy* est abandonnée. Le *proxy* joue ici le rôle d'un relais car il y a un *mapping* entre les *subflows* arrivants au *proxy* et ceux qui en sortent ;
2. **hôte distant ne supporte pas MPTCP** : le *proxy* fonctionne comme un *proxy* normal et non plus comme un relais de paquets. En effet, il doit gérer une fenêtre de congestion pour chaque *subflow* le liant à l'hôte mobile et une fenêtre également pour la connexion vers le serveur. Cela implique également que les paquets envoyés vers le serveur doivent être rassemblés dans l'ordre avant d'être envoyés sur le seul lien le connectant vers le serveur. Contrairement aux paquets faisant le chemin inverse qui doivent, eux, seulement être distribués sur les différents liens actifs en fonction de leur fenêtre de congestion.

2.6 Implémentation du kernel Linux de l'UCL

Le développement de l'implémentation du kernel Linux de MPTCP à l'Université Catholique de Louvain [32] a débuté en 2011 et est disponible à ce jour en version v0.92.

À l'instar de TCP, MPTCP est également implémenté dans le *kernel* de Linux. Il est donc nécessaire de le sélectionner au moment du démarrage de la machine dans le *bootloader* (e.g. GRUB).

2.6.1 Routage

Utiliser Multipath TCP sur plusieurs interfaces signifie que le *kernel* doit gérer plusieurs adresses IP simultanément. Pour permettre à celui-ci de les manipuler correctement, il est requis de lui indiquer : "Si je choisis telle adresse IP Source, utilise cette interface et ce *gateway* à la place de ceux par défaut." [32].

Cette opération est réalisée en créant une table de routage par interface. Habituellement le routage est guidé par des décisions

basées sur l'adresse de destination. Cependant il existe des cas pour lesquels ce paramètre n'est pas suffisant [30]. MPTCP fait partie de ces exceptions. Pour signaler au *kernel* de se baser également sur l'adresse source, nous devons mettre en place des politiques de routage indiquant : "Pour telle IP source, aller voir dans la table x" [32]. La configuration s'effectue avec le programme *ip rule* contenu dans le package `iproute2` [15].

Le processus entrepris alors par le *kernel* se déroule en deux étapes [32]. D'abord, il consulte la table des polices qui lui indique quelle table de routage consulter. La table correspondante est alors examinée pour obtenir le *gateway* spécifique à cette interface.

Lors de la création d'un *subflow* initial, le *kernel* va se baser sur la route par défaut de la table de routage générale pour décider sur qu'elle interface le créer.

La configuration de ces routes peut être réalisée de manière statique mais il est également possible de l'automatiser. Les scripts `mptcp_up` et `mptcp_down` disponibles sur le site internet de l'implémentation [32] permettent de réaliser cette étape.

2.6.2 *Schedulers*

Une fois les interfaces et le routage correctement configurés, une des décisions les plus importantes du *kernel* est de décider sur base de certains critères sur quel *subflow* il va devoir envoyer les données. Cette décision est très importante car elle a un impact sur la performance et l'expérience utilisateur. [6]

Le *scheduler* par défaut choisit de préférence le chemin avec le `RTT` le plus bas. Puisque le `RTT` est généralement plus faible sur une interface Ethernet qu'une interface Wi-Fi ainsi que le Wi-Fi par rapport à la 4G, ce *scheduler* aura principalement tendance à diriger les paquets sur l'interface Ethernet ensuite sur celle gérant le Wi-Fi. L'interface 4G ne sera quant à elle utilisée que dans le cas où une interface Wi-Fi ou Ethernet serait désactivée. [6]

Il existe également deux autres *schedulers* dans l'implémentation. Le *scheduler* Round Robin sélectionne les *subflows* l'un après l'autre afin d'utiliser le maximum de leur capacité. L'avantage de celui-ci étant que les données sont distribuées de manière équitable sur tous les *subflows*[25].

Le dernier *scheduler* proposé, appelé *redundant*, est un peu particulier puisqu'il transmet la même information sur toutes les interfaces disponibles. L'objectif étant d'obtenir la latence la plus basse possible, au détriment du *bandwidth*. [32]

2.6.3 *Fail-over*

Lorsque les données sont transmises sur les différents liens, il est possible qu'une interface tombe en panne, qu'un Wi-Fi devienne hors de portée lors du passage d'une borne Wi-Fi à l'autre (*handover*), etc. Tous ces cas sont pris en compte par le mécanisme de Fail-over. Il permet au kernel de réinjecter les paquets non acquittés sur une autre interface. [1] Le choix de l'interface est réalisé par le *scheduler*, qui est expliqué au point 2.6.2.

Le *kernel* dispose de deux moyens pour détecter la perte d'une interface :

1. soit l'adresse IP a disparu de l'interface ou alors celle-ci a été désactivée ;
2. soit le `Retransmission Time Out` (RTO) a expiré.

Le premier moyen de détection est immédiat alors que le second peut prendre jusqu'à plusieurs dizaines de secondes, en fonction du paramétrage.

Chapitre 3

Dispositif expérimental

3.1 Contexte

L'objectif de ce mémoire est de mesurer la capacité du protocole MPTCP à fournir une connexion internet aux terminaux mobiles dans le cas de déplacements en transport sur voie ferrée.

Des travaux réalisés sur MPTCP en transport sur voie ferrée ont déjà été réalisés lors du mémoire de TZEVELECOS [36]. Son objectif était de mesurer la combinaison de DASH (Dynamic Adaptive Streaming over HTTP) et de MPTCP lors de *streaming* vidéo. Il a notamment réalisé un test dans un trajet en train où il utilisait une interface en 3G et une interface Wi-Fi. Ce test a montré une forte variation du nombre de kilobits par secondes sur l'interface 3G. Il explique également que l'interface Wi-Fi n'a servi que dans la première gare, lors du lancement de la lecture du *streaming*. Internet n'était pas accessible dans les autres gares en raison de l'impossibilité d'accéder à la page d'authentification des portails. Nous tentons dans ce mémoire de réitérer l'expérience avec du *streaming* audio afin de comprendre pourquoi la page du portail était inaccessible. Nous discutons de ce point plus en détail dans la partie 6.4.1 du chapitre 6.

L'article [29] de RAICIU et al. s'intéresse à la mobilité avec MPTCP concernant une interface Wi-Fi et une interface 3G. Ils ont calculé une performance de 10-15% à l'avantage de MPTCP contre un TCP optimal utilisant le Mobile IP dans le cas d'un scénario de déplacement à pied et d'un autre véhiculaire à 60 km/h. Pour le déplacement véhiculaire, ils ont simulé un réseau Wi-Fi accessible pendant une durée moyenne de 13 secondes à un véhicule passant à proximité. Nous nous inspirons de cette expérience et la reproduisons dans le contexte des métros en générant un point d'accès Wi-Fi avec un smartphone. Pour le déplacement à pied, ils se sont basés sur des hotspots publics pour la connexion Wi-Fi. Nous envisageons également dans notre mémoire, l'utilisation de ceux-ci mais dans des environnements totalement différents que sont les transports sur rail.

XIAO et al. ont entrepris des scénarii sur base du protocole TCP muni d'une unique interface 3G dans des trains à grande vitesse (TGV) [39]. Ils ont comparé le comportement de TCP dans un contexte de très grande vitesse (plus de 300 km/h) avec un environnement moins rapide (100 km/h). Bien que le RTT et le *throughput* s'aggravent à très grande vitesse et soient très variables, ce dernier reste néanmoins décent. Dans notre mémoire nous ne considérons pas les TGV mais nous comparons également différents milieux et contrairement à cette étude nous l'envisageons dans un contexte de *multihoming* où MPTCP nous permet également de profiter d'une connexion par intermittence à des réseaux Wi-Fi.

DE CONINCK et BAERTS ont réalisé un mémoire dans lequel ils ont, notamment, mis en place un scénario de *streaming* audio. Celui-ci consistait à parcourir un trajet à pied muni d'un smartphone fonctionnant avec une version de MPTCP. Leur smartphone utilisait les interfaces 4G et Wi-Fi. Cette dernière interface se connectait aux hotspots publics disponibles le long du parcours. Ils concluent cette partie de leur rapport en affirmant que le *handover* est bien supporté par MPTCP et qu'il n'y a pas eu de coupure dans le *streaming* audio. Cependant ils montrent également que l'interface cellulaire est privilégiée de par sa stabilité au détriment de l'interface Wi-Fi alors que pour l'utilisateur cette dernière est plus intéressante au point de vue financier. Dans notre mémoire, nous poursuivons les investigations du *streaming* audio avec MPTCP dans un environnement de transport sur voie ferrée afin d'observer son comportement. Cependant, nous n'avons pas choisi le même dispositif expérimental comme mentionné dans ce présent chapitre et justifié au point 6.1 du chapitre 6.

Notre mémoire met principalement en place un scénario permettant de tester la continuité du réseau ainsi que sa performance lors d'accrochage de réseaux Wi-Fi. Pour ce faire, des tests ont été envisagés dans différents environnements de transports sur rail : dans les trains, trams et métros. Pour ces tests, il nous a semblé judicieux de réaliser du *streaming* audio car peu coûteux en terme de quantité de données et donc *in fine* en terme de coût sur la 4G mais également parce qu'il nous procure un flux continu.

Le scénario du VoIP n'a pas été envisagé dans ce mémoire étant donné que le protocole Real Time Transport (RTP) est plus généralement utilisé pour ce type d'application. De plus, RTP propose maintenant également une extension multipath nommée Multipath RTP (MPRTP) [2]. Cette extension fonctionne de manière similaire à MPTCP en divisant le flux RTP en sous-flux RTP et est rétro compatible avec RTP. [5]

3.2 Architecture

Pour réaliser nos mesures, nous avons mis en place une architecture où un client disposant du MPTCP communique avec un serveur MPTCP au moyen de deux interfaces : une interface Wi-Fi et une interface 4G, comme présenté sur le schéma de la figure 3.1.

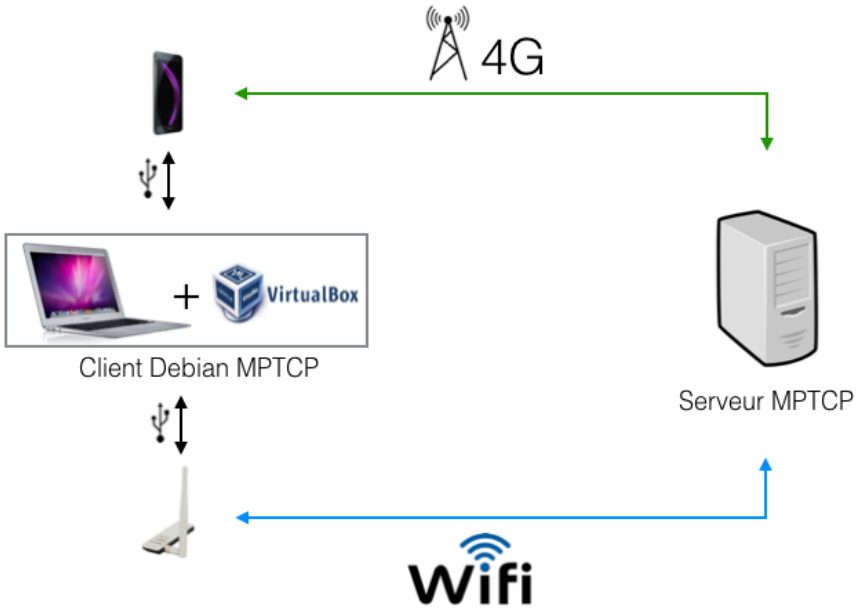

FIGURE 3.1 – Infrastructure pour le streaming audio

3.2.1 Serveur

Le serveur est hébergé par l'Université de Namur. Il tourne sous Debian GNU/Linux 8.8 (jessie) x64 et possède un unique lien réseau à 1000Mb/s sur lequel vont s'établir les deux *subflows* MPTCP. L'implémentation MPTCP de l'Université Catholique de Louvain (UCL) [32] en version v0.92 y est installée (version du kernel : 4.4.70.mptcp). Bien que l'adresse IP du serveur soit statique, le routage est configuré automatiquement lors de l'activation de l'interface Ethernet grâce aux scripts *mptcp_up* et *mptcp_down* disponibles sur le dépot github de l'implémentation [1].

Sur cette machine est installé Icecast 2 [40], un serveur de flux audio et vidéo, qui peut être utilisé entre autres comme station radio Internet. La version 2 de Ices [41] est utilisée conjointement. Cette dernière permet de lire des musiques au format Ogg Vorbis et de générer un flux audio envoyé à Icecast. Icecast s'occupe alors d'en faire un *broadcast* à destination des clients.

1. https://github.com/multipath-tcp/mptcp-scripts/tree/master/scripts/rt_table

Nous avons choisi Icecast 2 pour disposer d'un environnement similaire à celui employé lors des tests de *streaming* du mémoire [10] de DE CONINCK et BAERTS et ainsi mettre en parallèle leurs résultats avec les nôtres. De plus, l'utilisation d'un serveur de *streaming*, nous a permis d'initier une connexion directe entre le client et le serveur. En effet, si nous avions utilisé un service existant, e.g. Spotify ou une station radio Internet, nous aurions dû mettre en place un serveur *proxy* MPTCP. Dans ce cas, nous aurions rajouté un saut supplémentaire pour le transfert des paquets, ce qui aurait pu pénaliser le débit.

3.2.2 Client

Le client tourne dans une machine virtuelle Virtualbox sous Debian GNU/Linux 8.9 (jessie) x64. L'implémentation de l'UCL [32] de MPTCP en version v0.92 y est installée (version du kernel : 4.4.70.mptcp).

En guise de lien Wi-Fi, un adaptateur USB TP-Link TL-WN722N est utilisé. Il dispose d'une antenne à gain élevé de 4dBi, une vitesse Wi-Fi allant jusqu'à 150 Mbps et est compatible avec les normes IEEE 802.11b/g/n sur la fréquence 2.4GHz.

Pour la connexion au réseau mobile 4G, un smartphone HONOR 8 connecté sur le réseau mobile Orange est relié par USB à la machine virtuelle afin de l'utiliser en *tethering* [2]. Le combiné tourne sous Android 7.0 Nougat

Le réseau Orange (anciennement Mobistar) a été choisi car nous avons remarqué que le MPTCP ne passe pas sur certains réseaux mobiles de téléphonie belge, notamment Proximus [3] et Base.

Cette interface est considérée comme étant l'interface principale sur laquelle la connexion s'effectue. L'interface Wi-Fi est considérée comme une interface de soutien pour augmenter la bande passante du périphérique.

Dans un environnement statique, sans déplacement, le *round-trip time* (RTT) entre le client et le serveur est en moyenne de 23.4ms en passant par le Wi-Fi. En utilisant le lien 4G il grimpe alors à 81.7ms. Étant donné que l'on utilise le *scheduler* par défaut, l'interface Wi-Fi sera favorisée pour l'envoi des données grâce à son `RTT`, comme expliqué au point 2.6.2 du chapitre 2.

2. Le *tethering* est une technique qui permet de partager la connexion Internet avec un autre appareil.

3. Cette information a été confirmée par des contacts pris par ailleurs au sein de l'équipe de développement de l'implémentation du Kernel Linux de l'UCL malgré l'utilisation de la 4G chez Proximus lors du mémoire de DE CONINCK et BAERTS [10]

3.2.3 Smartphone Emporia

Lors de certaines mesures nous avons également eu besoin d'un smartphone Emporia pour créer des hotspots Wi-Fi. Ce smartphone tourne sous la version 4.4.2 Android KitKat.

Ce smartphone dispose d'un lien 3G relié au réseau Orange, la 4G n'étant pas compatible avec cet appareil. Le hotspot Wi-Fi généré permet le partage Internet de cette connexion 3G.

Chapitre 4

Analyse de l'environnement

4.1 Contexte

Avant de réaliser les tests de *streaming*, explorons les trois types d'environnement dans lequel les mesures prennent place. Les 3 premières parties de ce chapitre se concentrent chacune sur un environnement bien précis : les trains, les trams et les métros. Chacune d'entre elles présente les points d'accès Wi-Fi rencontrés et la couverture en réseau mobile. Enfin, la dernière partie analyse les réseaux Wi-Fi dans un environnement neutre, sans déplacement ni obstacle, pour comparer les résultats avec ceux obtenus dans les différents environnements.

Afin de pouvoir déterminer à quoi correspond de manière plus concrète les valeurs en dBm [1] pour le signal Wi-Fi de ce chapitre, nous définissons une échelle, présentée à la figure 4.1. Bien que cette échelle soit subjective, elle va nous permettre de qualifier la qualité du signal des points d'accès.

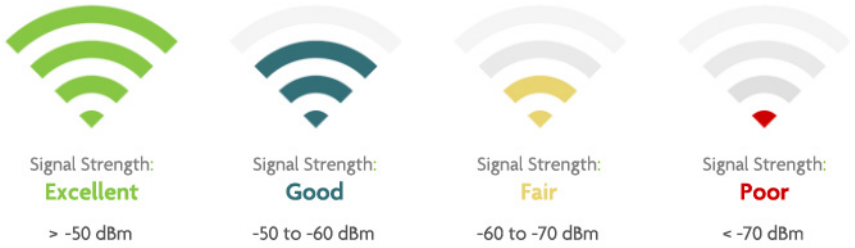

FIGURE 4.1 – Échelle de référence des dBm pour ce chapitre (Source : [22])

1. Le dBm indique la puissance du signal Wi-Fi et est exprimé en décibel par milliwatt

4.2 Trains

La première idée de ce mémoire était de réaliser des tests avec le MPTCP dans le train. L'objectif étant d'utiliser deux interfaces : la première étant connectée au réseau mobile et la seconde aux réseaux Wi-Fi. Notons que les tests sont effectués ici sur des lignes où la vitesse du train est inférieure à 160km/h [17], ce paramètre pouvant jouer un rôle lors de l'accrochage des réseaux Wi-Fi.

Pour la connexion 4G, la couverture réseau de l'opérateur Orange constitue 99.30% du territoire belge [16] selon l'Institut Belge des services Postaux et des Télécommunications (IBPT). Lors de nos mesures nous n'avons perdu la connexion que lors des passages dans les tunnels.

Pour la connexion Wi-Fi, la SNCB [2] met à disposition des bornes Wi-Fi spécifiques pour ses utilisateurs. En 2005 elle a installé, avec un opérateur téléphonique Belge TELENET, un réseau de hotspots Wi-Fi dans une cinquantaine de gares. Seulement une partie de ces hotspots est accessible pour la clientèle [3].

Pour obtenir une cartographie du réseau, les différents réseaux disponibles au cours de plusieurs trajets ont été scannés. L'objectif était de détecter les *Access Points* (APs) disponibles pendant le trajet et lors de nos passages en gare ainsi que l'intensité de leur signal.

Les graphes suivants illustrent la collecte des différents hotspots sur les lignes de Bruxelles-Central à Liège-Guillemins, de Bruxelles-Luxembourg à Namur, de Namur à Liège-Guillemins et de Genval à Louvain-La-Neuve.

La figure 4.2 présente les résultats du *scan* pour le trajet de Bruxelles-Central à Liège-Guillemins. Nous constatons que les APs disponibles se trouvent en début de trajet jusque Leuven (zone de Bruxelles) ainsi qu'en fin de trajet (zone de Liège) avec une grande zone de "no man's land". Cette zone s'explique par le fait que ce tronçon se situe sur une ligne à grande vitesse en plein milieu des champs.

Trois principaux opérateurs se partagent les différents APs rencontrés : TELENET, VOO et Proximus. Alors que les *hotspots* de TELENET et ceux de Proximus sont répartis équitablement sur la zone de Bruxelles et celle de Liège, les APs VOO ne sont disponibles que dans la partie wallonne du trajet.

2. La SNCB est une entreprise publique qui exploite le réseau ferroviaire belge.
3. Information récoltée auprès du service clientèle de la SNCB

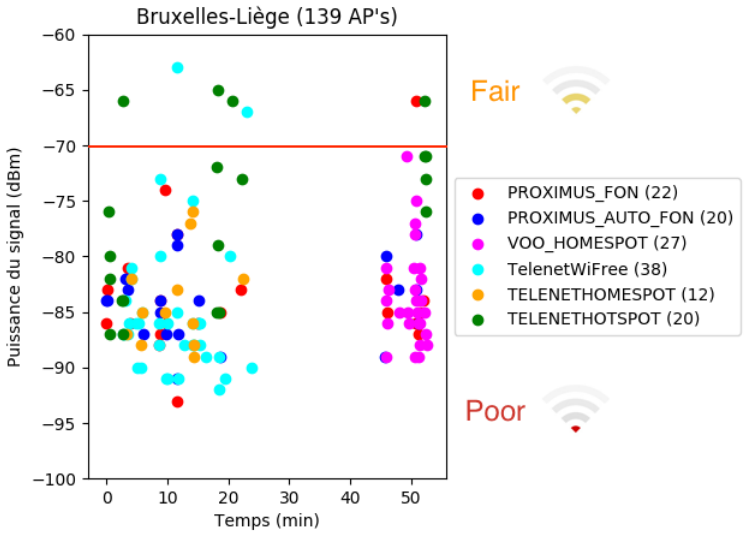

FIGURE 4.2 – APs disponibles sur le tronçon Bruxelles-Central - Liège-Guillemins

Les réseaux de Proximus n'étant pas directement installés en gare, les *hotspots* détectés sont généralement produits par des *boxes* internet domestiques. Un réseau `PROXIMUS_FON` et `PROXIMUS_AUTO_FON` sont en effet générés par défaut sur chacune des bornes récentes fournie par Proximus. Ces réseaux sont visibles mais il est impossible de s'y connecter lorsque le train est en marche, le temps passé à proximité étant trop court pour pouvoir "accrocher" le réseau. Ceux accessibles en gare sont quant à eux généralement situés trop loin du train. En effet, nous remarquons sur le graphe que la majorité des points d'accès Proximus a un niveau de signal en dessous de la barre des -80 dBm, ce qui est beaucoup trop faible, selon notre échelle définie au point 4.1, pour pouvoir s'y connecter. Enfin, le signal retenu sur le graphe correspond au meilleur signal rencontré pour chacun des points d'accès. En d'autres termes, il ne rend pas compte de la variation du signal. La moyenne du signal sera donc en pratique plus faible que ces maxima.

Les réseaux de VOO sont également générés par des modems domestiques avec le SSID[4] `VOO_HOMESPOT` ainsi que `TELENETHOMESPOT` pour les box TELENET. Les mêmes remarques que pour les bornes Proximus sont d'application et il nous a été impossible de nous connecter à ces différents points d'accès.

Enfin, le réseau `TELENETHOTSPOT` généré par des bornes TELENET a un signal maximum variant généralement de -65 dBm à -75 dBm qui correspond à une interprétation allant de "acceptable"

4. Le SSID est le nom du réseau

à "faible" sur notre échelle du point 4.1. Il est dès lors possible de s'y connecter lors du passage en gare, pour autant que les informations d'authentifications soient rapidement introduites sur le portail d'accès [5].

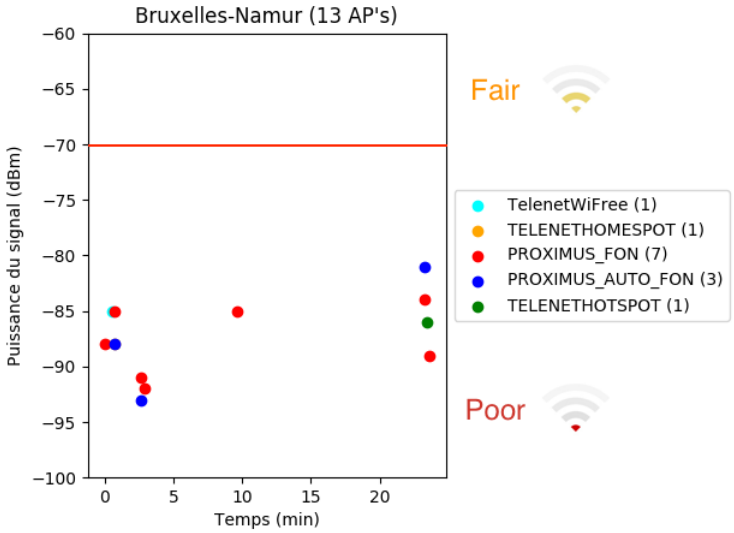

FIGURE 4.3 – APs disponibles sur le tronçon Bruxelles-Luxembourg - Namur

La figure 4.3, représentant les points d'accès sur le tronçon Bruxelles-Luxembourg/Namur, montre que trop peu de bornes sont disponibles lors du trajet. Le signal mesuré est inférieur à -80 dBm (niveau faible) ce qui rend impossible toute éventuelle connexion. Nous remarquons cependant l'apparition d'un nouveau SSID de TELENET : `TelenetWiFree`

Pour le trajet de Namur à Liège-Guillemins de la figure 4.4, nous remarquons une grande quantité de points d'accès par rapport aux graphes précédents. Ceci s'explique par le choix d'un omnibus pour faire le trajet plutôt qu'un train direct. Le train fait plus d'arrêts en gare ce qui permet de rester plus longtemps à côté de points d'accès que nous n'aurions probablement pas détecté en prenant le train direct.

Malgré l'abondance des points d'accès de Proximus et VOO, il est à nouveau très difficile de pouvoir s'y connecter. Les seuls points d'accès fiables sont les bornes `TELENETHOTSPOT`, quand elles sont disponibles lors d'arrêt en gare.

5. Ce point est discuté plus en détail dans la partie 6.4.1 du chapitre 6

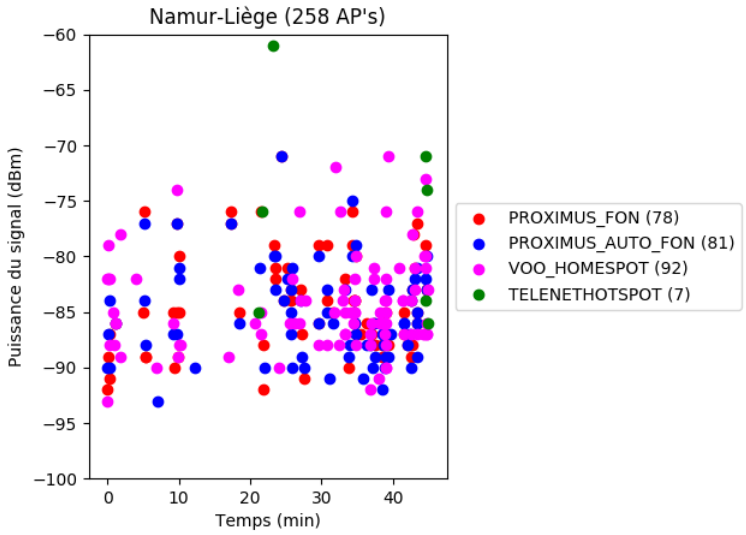

FIGURE 4.4 – APs disponibles sur le tronçon Namur - Liège-Guillemins

Enfin, le dernier trajet partant de Genval et allant jusque Louvain-La-Neuve représenté à la figure 4.5 n'apporte pas beaucoup plus d'informations : il n'a pas été possible de se connecter sur un point d'accès et aucun réseau de TELENET n'a été détecté.

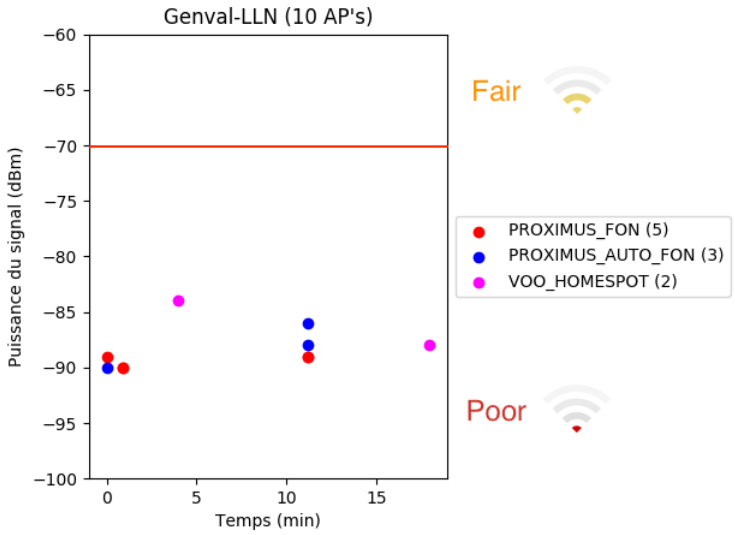

FIGURE 4.5 – APs disponibles sur le tronçon Genval - Louvain-la-neuve

En résumé, nous avons observé qu'un grand nombre de points d'accès proviennent des opérateurs TELENET, VOO et Proximus.

Pendant que le train est en marche, il est clairement impossible d'accrocher une borne Wi-Fi. Les APs qui nous intéressent sont ceux dont le temps de disponibilité est suffisant et dont le signal est suffisamment fort pour pouvoir s'y connecter. Les seuls points d'accès répondant à ces critères sont les hotspots de TELENET se trouvant dans certains points d'arrêts.

4.3 Trams

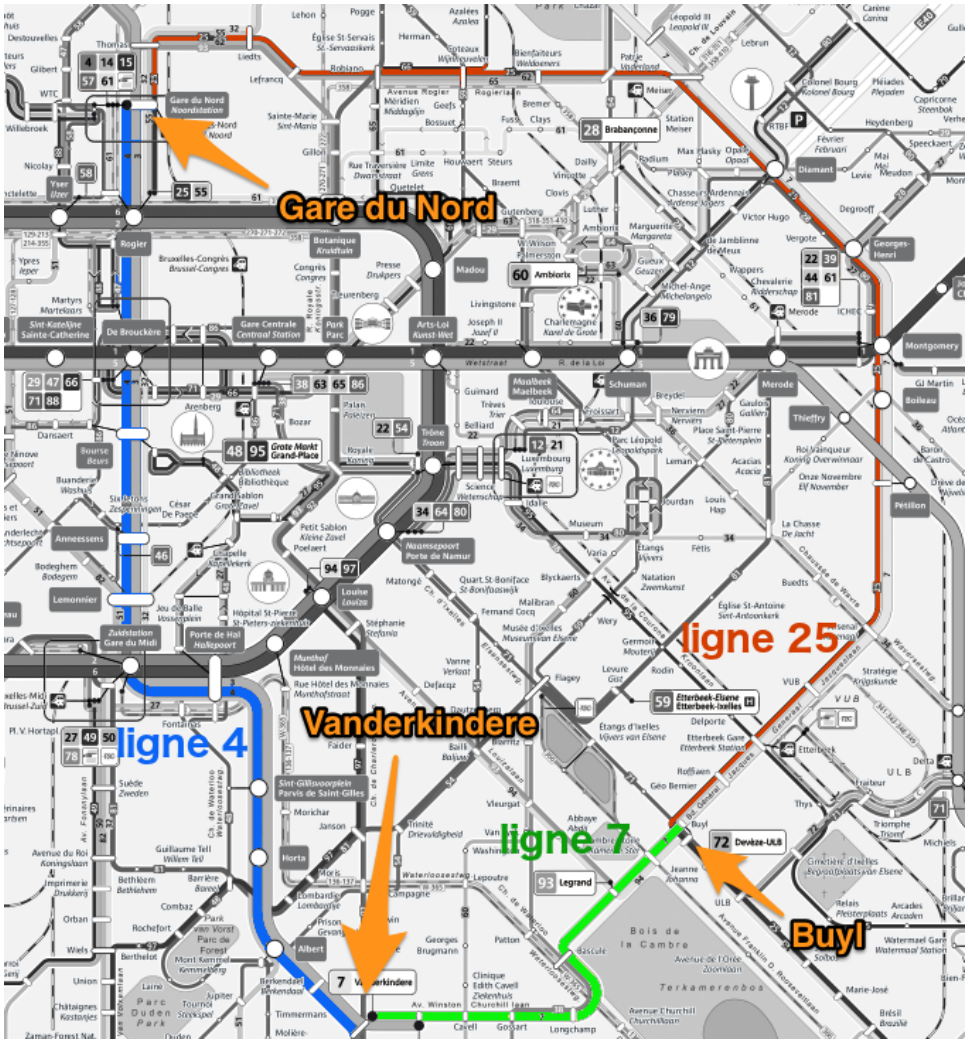

FIGURE 4.6 – Carte des 3 lignes de Tram de Bruxelles analysées

Dans la partie précédente, nous avons conclu que l'utilisation de MPTCP avec un lien 4G et un lien Wi-Fi était possible dans le train à condition d'avoir accès aux bornes de TELENET mises à disposition par la SNCB dans certaines gares. Mais qu'en est-il du MPTCP dans les trams ? Est-il possible de bénéficier de deux interfaces en

permanence ? Y a-t-il suffisamment de points d'accès pour rendre l'utilisation de MPTCP utile ? Cette partie va tenter de répondre à ces questions.

Pour se rendre compte de la situation, un circuit en tram passant par 3 lignes de la capitale belge a été envisagé. Ce trajet est représenté à la figure 4.6. Les graphes 4.7, 4.8 et 4.9 présentent respectivement les hotspots disponibles le long de la ligne 25 entre Buyl et la gare du Nord, de la ligne 4 entre la gare du Nord et Vanderkindere et à proximité de la ligne 7 partant de Vanderkindere à l'arrêt Buyl.

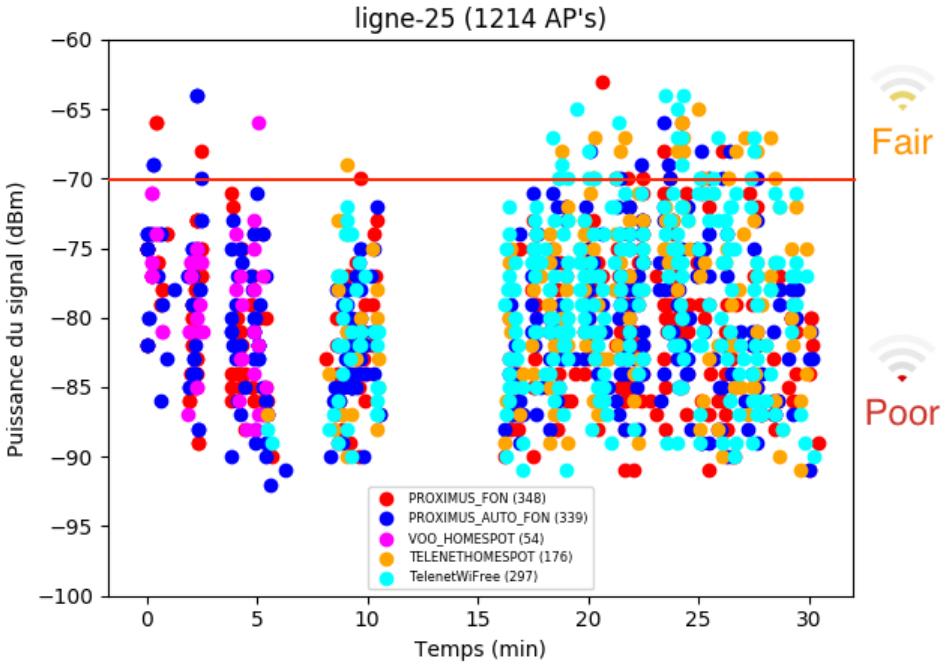

FIGURE 4.7 – APs disponibles sur le tronçon Buyl - Gare du Nord de la ligne 25

Le graphe 4.7 nous indique, sans grande surprise, que le nombre de points d'accès disponibles est environ 5 fois plus grand que pour le tronçon Namur-Liège en train pour un trajet presque deux fois moins long. Ce n'est pas surprenant étant donné le nombre d'habitations proches du passage du tram disposant d'une box générant un hotspot public. Sur ce même graphe, nous observons un espace sans point d'accès entre la minute 6 et 8, qui peut s'expliquer par le fait que cette portion du trajet en site propre est un peu plus éloignée des habitations. Un second passage à vide entre la minute 12 et 16 correspond à la descente du tram dans un sous-terrain reliant plusieurs stations.

Contrairement aux points d'accès disponibles dans les trains, ceux accessibles en trams ont globalement un meilleur signal. Alors que le signal dans les trains ne parvenait pas à dépasser la barre des -80 dBm, dans les cas des trams, le signal est plus ou moins équitablement réparti entre -70 dBm et -90 dBm. C'est une amélioration, certes, mais cette valeur reste relativement faible par rapport à notre échelle du point 4.1.

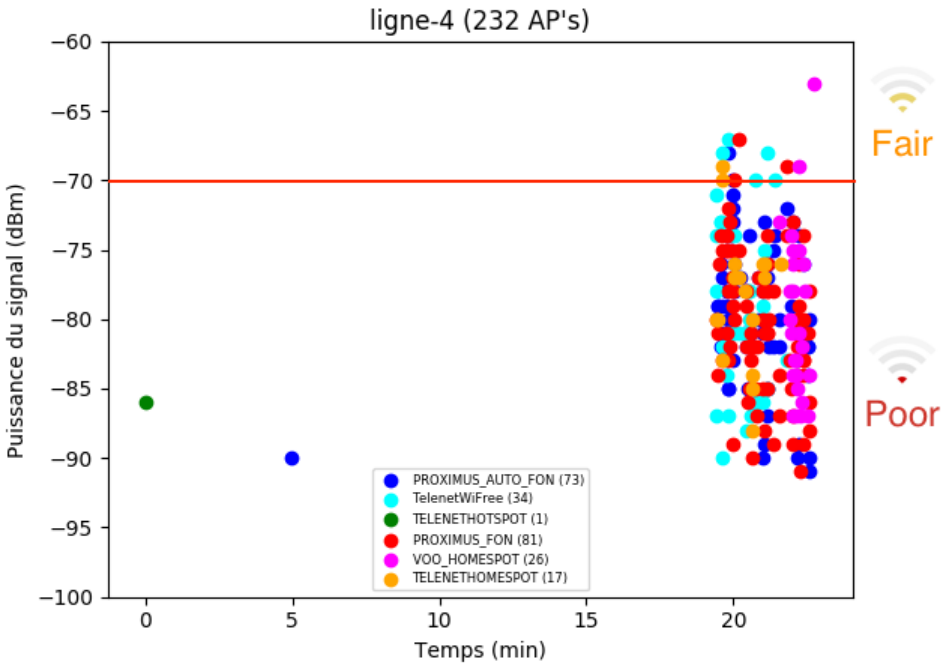

FIGURE 4.8 – APs disponibles sur le tronçon Gare du Nord - Vanderkindere de la ligne 4

Sur la deuxième ligne parcourue, exposée à la figure 4.8, il faut attendre la 20ème minute pour obtenir des réseaux intéressants. En effet, cette ligne traverse le centre de Bruxelles du Nord au Sud et est par conséquent en sous-sol jusqu'à la vingtième minute à Bruxelles-midi où elle continue son chemin en surface jusqu'à l'arrêt Vanderkindere.

La STIB[6] ne prévoit pas de hotspots particuliers pour les passages sous-terrains de même que pour les lignes en surface lors de passage en station. L'utilisation du MPTCP est donc peu utile sur ce tronçon.

6. La STIB est un organisme belge de transport public

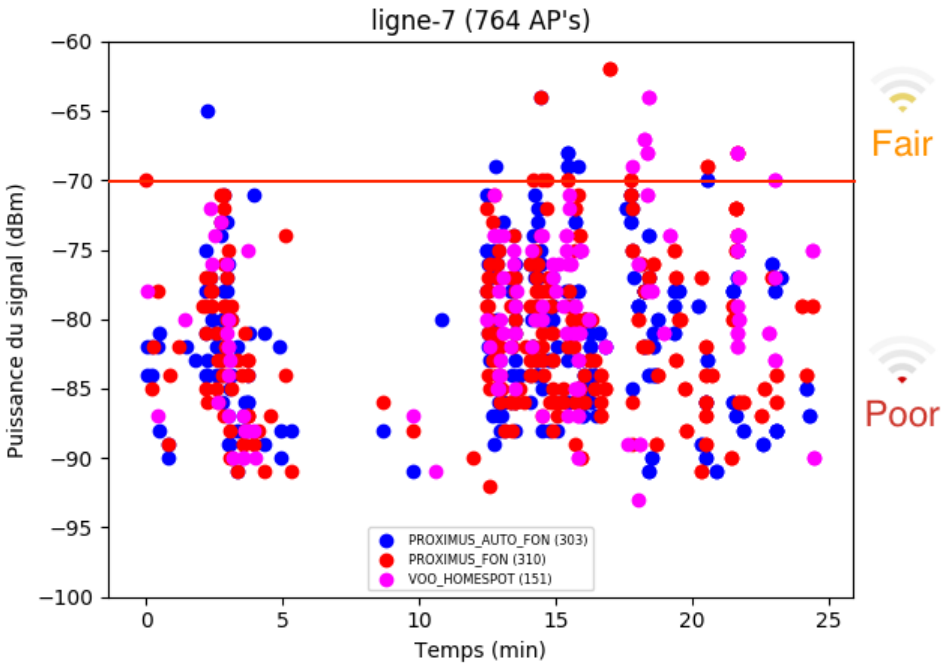

FIGURE 4.9 – APs disponibles sur le tronçon Vanderkindere - Buyl de la ligne 7

Enfin la figure 4.9 présente les points d'accès de la ligne 7. Les résultats sont similaires aux données récoltées sur la ligne 25, si ce n'est qu'aucun hotspot de TELENET, pourtant bien présent sur la première ligne visitée (au nombre de 473), n'est disponible sur ce tronçon.

Pour conclure cette partie, l'analyse des points d'accès présents lors des trajets en tram a montré la pertinence de l'utilisation de MPTCP dans un tel environnement. Sur les trois lignes de tram parcourues, plus de 1800 hotspots publics ont été détectés pendant environ 1h et demie de trajet, ce qui est plutôt encourageant. Néanmoins il faut prendre en considération le fait que plus de la moitié de ces accès ne sont disponibles que pendant un laps de temps limité et que les trajets sous-terrains ne comportent pas de points d'accès disposant d'un signal suffisamment fort selon notre échelle définie au point 4.1.

4.4 Métros

Pour le métro, les points d'accès Proximus, VOO et TELENET sont visibles, comme le montre la figure 4.10, mais en pratique il n'est pas possible de s'y connecter.

Nous remarquons en revanche la détection d'un nouveau SSID : `wifi.brussels`. Ces hotspots sont actuellement mis en place par la STIB dans les stations de métros. Il est prévu que tous les arrêts seront dotés de ce réseau Wi-Fi d'ici fin 2017 [8].

Cet environnement semble donc propice pour l'utilisation du MPTCP. Le réseau sous-terrain disposant également d'une connexion 4G permanente, le dispositif pourrait se connecter aux bornes `wifi.brussels` en entrant dans les stations tout en ayant une interface avec un réseau 4G continu tout au long du trajet.

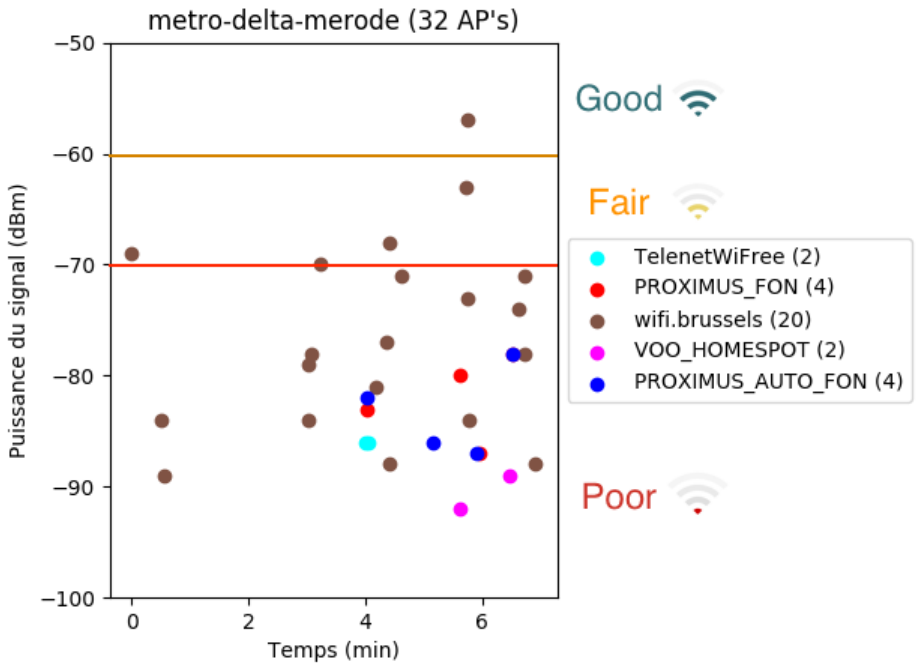

FIGURE 4.10 – APs disponibles sur le tronçon Delta - Merode de la ligne 5 de métro

Le gestionnaire du réseau `wifi.brussels` met également en place une fonctionnalité permettant de rester connecté d'un point d'accès à l'autre en mémorisant l'adresse MAC [18]. Malheureusement cette fonctionnalité est défaillante à l'heure ou nous écrivons

ces lignes [7].

De plus, malgré le fait que certains hotspots aient un signal "acceptable" voire "bon" selon l'échelle du point 4.1, nous n'avons pas réussi à nous connecter sur les bornes lors du trajet. Or avec un smartphone cette connexion est tout à fait possible. Cette différence est expliquée par une périodicité de balayage (*scan*) différente entre les deux périphériques, celle du dispositif expérimental étant insuffisante. Ce point est débattu en profondeur dans la partie 6.2.1 du chapitre 6.

4.5 Environnement neutre

Malgré l'abondance des hotspots Wi-Fi lors des trajets en trams, les connexions sont très rares et très aléatoires. Afin de mieux comprendre pourquoi la carte Wi-Fi a des difficultés à se connecter aux différents hotspots lors des trajets en tram et en train, des tests dans un environnement neutre ont été mis en place. Pour réaliser ces tests, une borne `PROXIMUS_AUTO_FON`, fonctionnant avec le protocole 802.1x et sécurisé avec WPA2 Enterprise, a été utilisée.

Ces mesures permettent de déterminer comment évolue le signal ainsi que le temps de connexion au fur et à mesure que l'on s'éloigne du point d'accès. Ces tests sont réalisés dans un environnement le plus idéal possible, sans murs, ni vitres, ni aucun objet pouvant altérer la puissance du signal.

L'objectif est de valider deux hypothèses. La première, bien qu'évidente, est de voir si le signal diminue plus la station (STA) s'éloigne du point d'accès, mais plus généralement de savoir jusqu'à quelle intensité du signal il est possible de se connecter. La deuxième hypothèse formulée est que le temps de connexion a tendance à augmenter plus la distance avec le point d'accès est grande.

Le graphe de la figure 4.11 présente les résultats obtenus. Pour les obtenir, un script a été réalisé. Il est lancé plus ou moins tous les 5 mètres sur une distance de 100 mètres. Étant donné la variabilité du signal Wi-Fi, il prend cinq fois la mesure et en fait la moyenne. La durée de connexion au hotspot est elle aussi effectuée cinq fois afin de détecter une éventuelle variabilité.

La première remarque que l'on peut formuler est que, comme attendu, plus nous nous éloignons du point d'accès, plus la puissance du signal diminue. Nous pouvons aller jusqu'à dire qu'il diminue

7. Le gestionnaire du réseau wifi.brussels a été contacté et tente toujours actuellement de résoudre le problème.

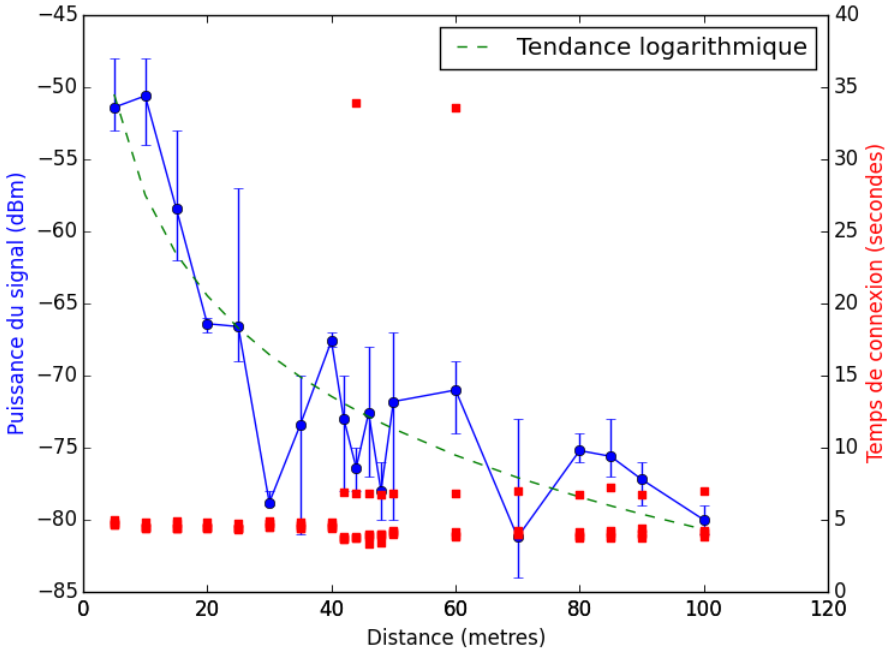

FIGURE 4.11 – Test du signal Wi-Fi d'une borne PROXIMUS_AUTO_FON

de manière logarithmique étant donné l'échelle des dBm. Ce que nous apprend également le graphe c'est que même à une distance de 100 mètres, la connexion peut malgré tout être établie.

En considérant maintenant le temps de connexion, nous observons, contrairement à notre seconde hypothèse, qu'il n'augmente pas en fonction de la distance mais qu'il reste constant bien au contraire. De manière générale, il nécessite +/- 4 secondes ou +/- 7 secondes pour se connecter. Cependant à 44 mètres, au cours d'une mesure, il prend un peu moins de 35 secondes et à 60 mètres, il prend également à deux reprises un peu moins de 35 secondes. Ce résultat peut être dû à un mouvement ou à un mauvais positionnement de l'antenne Wi-Fi ou à des interférences.

Nous pouvons donc déduire de ces mesures en environnement neutre que si la STA a des difficultés à se connecter aux hotspots dans les trams ce n'est vraisemblablement pas dû à une durée de connexion élevée mais plutôt à un signal beaucoup trop faible pour initier la connexion.

Cette conclusion ne peut malheureusement pas être validée à 100% étant donné que les mesures n'ont pas pu être établies au delà de 100 mètres. Nous ne pouvons dès lors pas garantir que le temps de connexion soit le même jusqu'à une distance où l'on ne capte

plus le signal.

Parallèlement à ce test, nous pouvons également noter que l'ajout d'un double vitrage induit une diminution de presque 10 dBm. Or, les signaux des Wi-Fi publics des particuliers parvenant aux trams doivent généralement traverser le mur extérieur des bâtiments pour ensuite traverser la vitre du tram. Nous pouvons donc imaginer la perte de puissance du signal arrivant au récepteur malgré l'intensité des signaux mesurée lors des trajets.

À cela s'ajoutent également les phénomènes de réflexion, réfraction, éparpillement et diffraction des ondes [11] qui, opérés sur les murs, panneaux de signalisations, arbres, objets mobiles tel que les voitures et les bus, ont également un impact sur la qualité du signal reçu.

L'environnement étant très complexe, il devient alors concevable d'accepter le fait que la STA n'ait pas réussi à se connecter lors des trajets en tram.

Chapitre 5

Streaming audio

5.1 Déroulement des mesures

Afin de tester les capacités de *handover* [1] du MPTCP dans les environnements de transport sur voie ferrée, nous avons mis en place des tests de *streaming* audio. Ils vont nous permettre de déterminer si l'utilisation d'un chemin Wi-Fi combiné à l'interface cellulaire va apporter un gain de débit par rapport à une connexion 4G seule.

Les mesures ont été réalisées dans les deux types d'environnement dans lesquels nous avons observé les points d'accès et dans lesquels il a été possible de se connecter lors du chapitre 4, soit en train et métro, afin de juger si MPTCP est capable d'apporter un meilleur débit lors de passages en station ou en gare.

5.2 *Streaming* audio dans le métro

5.2.1 Scénario

Étant donné qu'il n'est pas possible de se connecter lors du trajet sur les bornes `wifi.brussels` comme énoncé au point 4.4 du chapitre 4, un hotspot Wi-Fi a été généré avec un smartphone Emporia. Ce dernier est connecté à la 3G sur le réseau Orange et disposé en station sur le quai à une distance de 3 à 4 mètres des rails. Cet environnement artificiel permet de tester MPTCP dans les métros malgré les limitations du `network-manager`.

Étant donné que l'interface cellulaire du smartphone Emporia passe par la 3G et celle du dispositif expérimental par la 4G, il nous a semblé nécessaire de forcer la 3G également sur ce dernier. Cette opération permet de ne pas induire de biais dans les résultats en favorisant l'envoi de paquets sur l'interface cellulaire 4G alors que dans un scénario Wi-Fi - 4G, le Wi-Fi est à privilégier [2].

1. Les différents types de *handovers* sont présentés au point 2.3.6 du chapitre 2
2. le point 3.2.2 du chapitre 3 explique l'impact du `RTT` sur le *scheduler*

MPTCP est configuré en *fullmesh* (expliqué au point 2.3.6 du chapitre 2) afin de tirer parti des deux interfaces réseaux lorsqu'elles sont toutes les deux actives.

Le scénario se déroule comme suit :

1. le partage en *tethering* de la connexion 3G est activé et la carte Wi-Fi allumée à quai ;
2. une trace réseau est enregistrée sur les deux interfaces à l'aide de tshark [9] et la lecture du flux audio est lancée dans VLC Media Player [37] ;
3. l'ordinateur est amené dans la rame de métro avant qu'elle ne démarre ;
4. le métro s'arrête dans la station suivante à proximité du point d'accès. L'interface Wi-Fi se connecte sur le réseau généré par le smartphone Emporia et envoie une option `MP_JOIN` pour créer un sous-flux MPTCP ;
5. le métro continue son trajet jusqu'au prochain arrêt où la capture est arrêtée.

5.2.2 Analyse des résultats

Les résultats pour le *streaming* audio réalisés dans le métro montrent que le *subflow* de soutien sur l'interface Wi-Fi ne s'est créé que 2 fois sur les 4 tentatives réalisées. Ce taux de réussite est expliqué par le temps total de connexion au serveur de *streaming* audio.

Tout d'abord, il est nécessaire de savoir que le temps d'arrêt du métro en station est en moyenne de 15 à 20 secondes. Durant cet intervalle, la carte Wi-Fi doit, dans un premier temps, détecter le hotspot, se connecter et enfin créer un *subflow* MPTCP.

Le temps de détection du hotspot est très élevé et expliqué par le fait que l'intervalle de temps entre les *scans* de réseau sous Debian est trop long. Ce point est débattu en détail au point 6.2.1 du chapitre 6.

Une fois le hotspot détecté, le temps de connexion prend en général entre 4 et 7 secondes, comme indiqué au point 4.5 du chapitre 4. Enfin, la création du nouveau *subflow* MPTCP est peu significative étant donné qu'elle ne prend pas plus d'une seconde.

Nous observons donc qu'en utilisant le network-manager le temps total (détection Wi-Fi, connexion Wi-Fi et création du *subflow* MPTCP) consacré à la connexion au hotspot est très élevé. Cette valeur explique le faible taux de réussite de nos essais dans les métros.

Analysons maintenant les données récoltées lors de nos mesures fonctionnelles. Le schéma de la figure 5.1 nous montre que le débit en *bytes* par seconde reste constant sur l'interface 3G jusqu'à l'arrivée en station à la seconde 200 qui correspond au moment du *handover* [3] MPTCP. Ensuite cette valeur devient plus variable. Cependant lors d'une deuxième mesure réalisée sur le même tronçon, non reprise dans ce mémoire, nous n'avons pas observé cette fluctuation. Nous ne pouvons donc pas tirer de conclusion à propos à ce phénomène.

Ce qui est frappant également, c'est la faible utilisation de l'interface Wi-Fi. Effectivement, sur un court trajet de moins de 5 min (incluant le temps d'attente sur le quai), cette interface n'est disponible que pendant les 20 secondes d'arrêt du métro. Cependant cette valeur est à relativiser : si nous avions utilisé les hotspots disponibles dans les stations à la place du smartphone Emporia, nous aurions capté du réseau Wi-Fi en attendant le métro. De plus, le temps de transport entre les stations se situe entre 30 secondes et 1 minutes. Sur une trace dans l'environnement réel, le temps de transfert serait donc plus équilibré sur les deux interfaces.

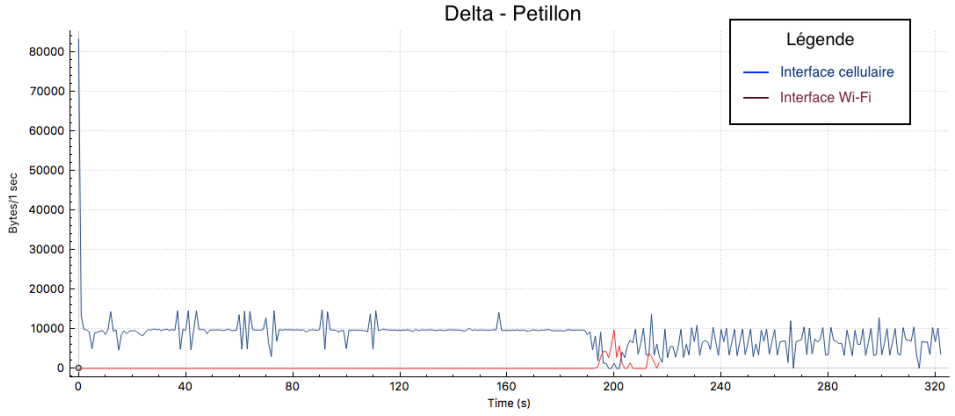

FIGURE 5.1 – Streaming audio dans le métro

Un agrandissement de la figure 5.1 représenté à la figure 5.2 se focalise sur le *handover* MPTCP. Nous y remarquons que l'interface cellulaire passe au second plan lorsque l'interface Wi-Fi transfère de l'information mais continue néanmoins à recevoir le flux audio.

Malgré la faiblesse de l'interface 3G au moment du *handover*, le flux audio reçu ne comporte néanmoins pas de signe de coupure. La figure 5.3 représente les données sonores récoltées de manière graphique.

3. Le fullmesh est un des types de *handover* proposés par l'implémentation de l'UCL, comme développé à la page 2.3.6 du chapitre 2

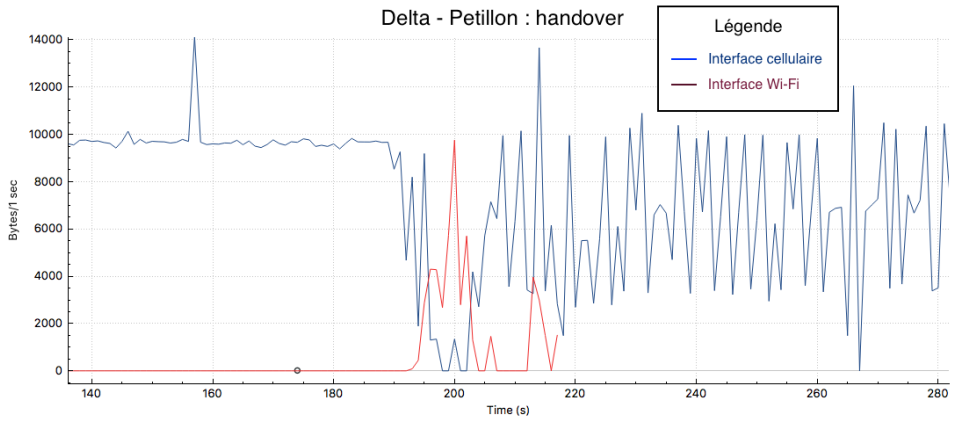

FIGURE 5.2 – Streaming audio dans le métro - handover

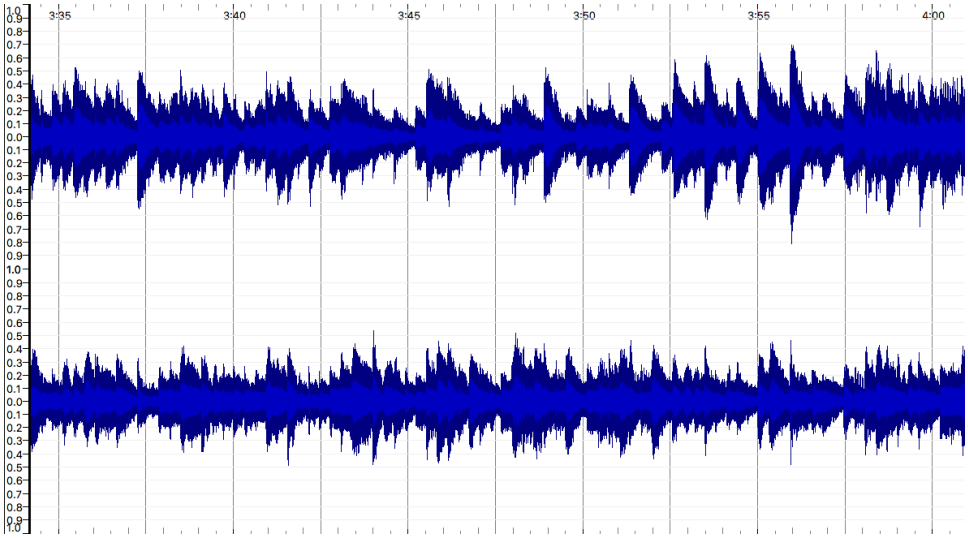

FIGURE 5.3 – Flux audio dans le métro

Nous pouvons conclure de ce premier test de *streaming* audio dans le métro que l'utilisation du MPTCP est possible et n'entraîne pas d'impact négatif sur l'expérience utilisateur au niveau de l'écoute du flux audio. Cependant l'avantage réel de mettre en place MPTCP dans un tel environnement semble *in fine* limité en terme d'amélioration du débit.

5.3 *Streaming* audio dans le train

5.3.1 Scénario

Le point 4.2 du chapitre 4 sur le Wi-Fi dans les trains nous a appris qu'il n'est pas possible de se connecter aux hotspots générés

par les *boxes* internet des particuliers. Par contre, les points d'accès `TELENETHOTSPOT` mis en place dans les gares sont quant à eux utilisables dans le cadre de nos mesures.

Le *path-manager* de MPTCP est configuré en *fullmesh* (expliqué au point 2.3.6 du chapitre 2) afin de tirer parti des deux interfaces réseaux lorsqu'elles sont actives simultanément.

Nous réalisons pour ce faire un trajet en train partant de Liège-Guillemins et arrivant à Bruxelles-Central en passant par Leuven et Bruxelles-Nord.

Le scénario se déroule comme suit :

1. le partage en *tethering* de la connexion 4G est activé et la carte Wi-Fi également ;
2. l'ordinateur est amené dans le train avant que ce dernier ne démarre ;
3. une trace réseau est enregistrée sur les deux interfaces à l'aide de tshark [9] et la lecture du flux audio est lancée dans VLC Media Player [37] quelques minutes avant d'entrer dans la gare suivante (Leuven) ;
4. le train s'arrête dans les deux gares suivantes. L'interface Wi-Fi se connecte sur le réseau disponible et envoie une option `MP_JOIN` pour créer un sous-flux MPTCP ;
5. le train continue son trajet jusqu'à l'arrêt suivant où la capture est arrêtée.

Lorsque nous arrivons dans la gare de Leuven deux cas de figure se présentent pour le Wi-Fi : soit le réseau a mémorisé l'adresse MAC et la connexion Internet est directement rétablie lors de la connexion au portail, soit elle ne l'a pas mémorisé et il est alors nécessaire de s'authentifier à nouveau.

Pour faire face à ce deuxième cas, il a été nécessaire de créer un script pour automatiser l'authentification sur le portail. Pour accéder à la page de connexion du portail, l'adresse IP de l'interface Wi-Fi a été mentionnée comme adresse IP source des paquets pour que les échanges s'effectuent bien sur cette interface et non sur l'interface 4G déjà connectée à Internet. Le problème d'authentification aux portail est expliqué plus en détails au point 6.4.1 du chapitre 6. Le code source du script est disponible en annexe au point 8.1.

5.3.2 Analyse des résultats

Lors du premier arrêt du train à Leuven, une connexion a été effectuée sur le point d'accès. Le temps d'arrêt prévu était de 3 min en théorie, en pratique il s'est avéré plus court. Cependant nous remarquons sur la figure 5.4 que le transfert audio s'est lancé sur l'interface Wi-Fi pendant une vingtaine de secondes.

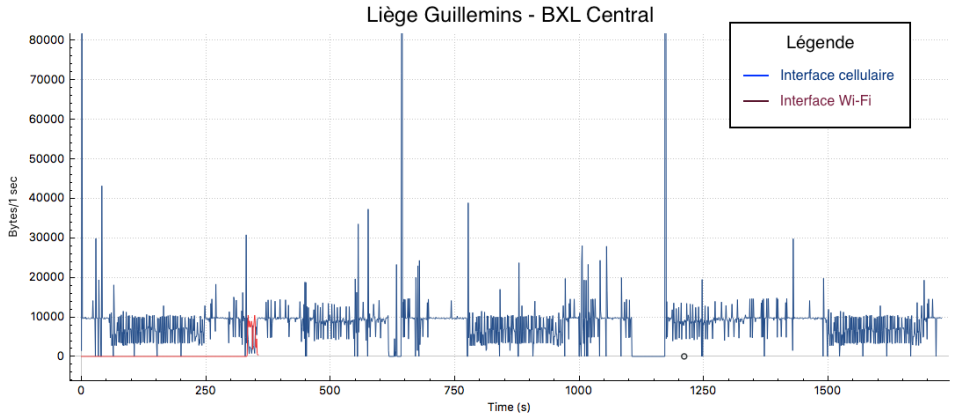

FIGURE 5.4 – Streaming audio dans le train

Nous observons également sur la figure 5.4 que le transfert sur l'interface 4G a été interrompu lors d'un passage sous un tunnel avant de reprendre dès la connexion rétablie à la sortie du tunnel. Le transfert a repris malgré l'utilisation du NAT lors du *tethering*. Le NAT ne pose pas de problème en cas de *handover* cellulaire alors qu'il en pose lors de *handover* Wi-Fi. Cette constatation est expliquée plus en détail au point 6.3.1 du chapitre 6.

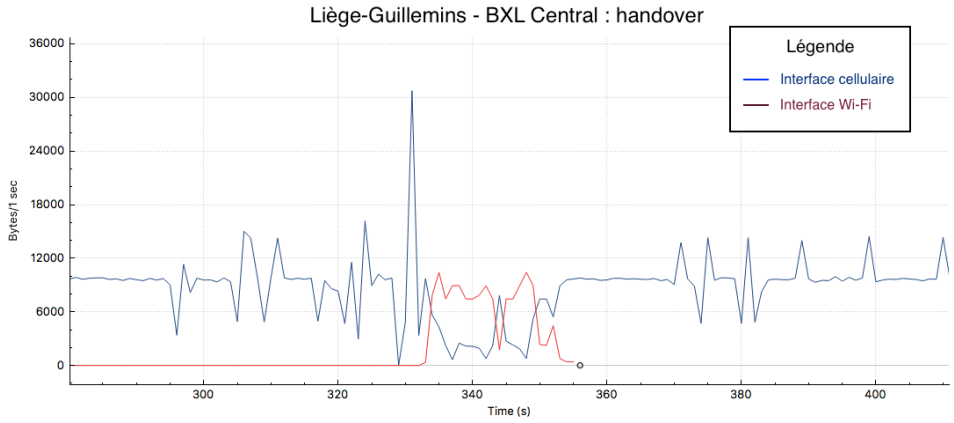

FIGURE 5.5 – Streaming audio dans le train - handover

L'agrandissement de la figure 5.4 représenté à la figure 5.5, nous montre plus en détail le *handover* MPTCP. Nous observons une chute du nombre de bytes par seconde sur l'interface cellulaire lorsque l'interface Wi-Fi est mise à contribution. Cette dernière étant configurée en *fullmesh* et non en *subflow* de *backup*, ce phénomène est plutôt surprenant au premier abord. Il peut cependant s'expliquer par la valeur du `RTT` du Wi-Fi. Cette dernière étant plus petite que le `RTT` de l'interface 4G, comme constaté dans un environnement statique au point 3.2.2, il démontre la décision du *kernel* à privilégier l'interface Wi-Fi au détriment de la 4G dès qu'elle apparaît.

Chapitre 6

Critique des mesures effectuées

Ce chapitre porte un regard critique sur les mesures effectuées afin de nuancer les résultats obtenus lors des mesures des chapitres précédents. Il justifie les choix architecturaux du dispositif expérimental, expose les problèmes rencontrés et propose des parades possibles.

6.1 Justification des choix architecturaux

Pour réaliser nos mesures, axées sur la mobilité, nous avions dans un premier temps envisagé de les effectuer sur smartphone. Cependant, la dernière version du *kernel* développée par l'Université Catholique de Louvain [32] est seulement disponible sur les smartphones Google Nexus. Ne disposant pas d'un tel équipement, nous nous sommes rabattus sur la dernière version du *kernel* disponible sous Debian. Nous l'avons alors installée dans une machine virtuelle pour plus de facilité (*snapshots*, restaurations, etc.).

Concernant l'interface Wi-Fi, nous avons opté pour une clé Wi-Fi externe. Bien qu'il aurait été possible de créer des adaptateurs virtuels configurés en Bridge ou en NAT, il comporte un inconvénient majeur : celui de partager la connexion Internet avec l'hôte. De plus, le NAT aurait entravé l'utilisation du MPTCP lors de *handover* Wi-Fi, comme expliqué en détail au point 6.3.1. Pour ces deux raisons, il nous a semblé préférable d'utiliser une interface Wi-Fi dédiée à notre machine virtuelle.

Pour la connexion cellulaire, il aurait été intéressant d'utiliser une clé 4G externe, l'utilisation d'un smartphone en *tethering* générant du NAT. Cependant dans le contexte du *handover* cellulaire, le NAT ne pose pas de problème avec MPTCP comme expliqué au point 6.3.1. De plus, étant donné le prix de ce périphérique, il nous a semblé acceptable de ne pas s'en procurer, bien que le partage de la connexion 4G entre le smartphone et la machine virtuelle puisse entraîner des biais dans les résultats.

6.2 Périodicité du scan

6.2.1 Problème

Dans les métros, nous avons observé que le dispositif expérimental n'arrivait pas à se connecter pendant les trajets sur les bornes `wifi.brussels` alors qu'un smartphone Android quelconque y arrive pourtant bien. Nous avons également décelé un souci au niveau de la détection des hotspots lors du *streaming* audio dans les trains où l'interface Wi-Fi n'arrivait pas à se connecter aux bornes `TELENETHOTSPOT` lors de certaines tentatives.

Tous ces problèmes sont liés à la périodicité du *scan* de réseau du network-manager [13], le gestionnaire de réseau par défaut sous Debian. Nous observons un temps de détection plutôt élevé, comme le montre le tableau de la figure 6.1.

Numéro du test	1	2	3	4	5	6	7	8	9	10
Durée (sec)	20	11	48	57	13	14	12	5	32	13

TABLE 6.1 – Mesures de la durée de détection d'un hotspot avec le network-manager

Ces valeurs ont été obtenues en calculant le temps que met le network-manager pour détecter un hotspot nouvellement créé à l'aide d'un smartphone. Nous avons généré 10 fois le point d'accès. La moyenne de détection du hotspot a été de 22,5 secondes.

En inspectant le code source [14] du network-manager, nous pouvons confirmer cette valeur expérimentale. En effet, dans le code, l'intervalle de temps entre les différents *scans* des réseaux Wi-Fi se situe entre 20 et 120 secondes. La valeur choisie étant probablement basée sur des heuristiques. Par exemple si le dispositif est déjà connecté à un réseau, il n'est pas nécessaire de rafraîchir la liste des réseaux disponibles rapidement.

D'après [34], la version Android Gingerbread est configurée avec un intervalle de 5 secondes par défaut. Cette valeur expliquerait pourquoi le smartphone peut se connecter sur le point d'accès quand le dispositif expérimental n'y arrive pas.

6.2.2 Solutions possibles

Une des solutions possibles résultant du débat mené au point précédent et venant immédiatement à l'esprit est dans un premier temps de vouloir configurer la valeur de rafraîchissement du *scan* de réseau. Cependant, il semblerait qu'il n'y ait aucun moyen de

configurer cette valeur autrement qu'en recompilant le programme network-manager [42]. À notre meilleure connaissance, cette opération n'a pas été testée. Par prudence, nous ne l'avons pas tentée car il faut prendre en compte les impacts de la modification d'une telle valeur.

Un intervalle trop petit pourrait avoir des implications au niveau de la consommation de la batterie ainsi qu'au niveau de la charge réseau des points d'accès adjacents. En effet, la STA utilise dans ce cas un balayage actif [1] qui consiste à les sonder en leur envoyant des requêtes. Ceux-ci répondent alors avec un *beacon* contenant leur SSID et leur adresse MAC. Un nombre élevé de ces requêtes entraîne une plus grande utilisation des ressources de la STA et génère également plus de trafic réseau.

Une deuxième solution serait de "forcer" une détection des réseaux toutes les secondes en ligne de commande. Cependant, probablement en lien avec les remarques du paragraphe précédent, le network-manager garde en mémoire les SSIDs obtenu lors du précédent *scan* de réseau à la place de détecter réellement les nouveaux réseaux. Nous obtenons donc exactement les mêmes résultats que ceux récoltés dans le tableau 6.1 du point 6.2.1.

La troisième possibilité serait de forcer la connexion au hotspot toutes les secondes en ligne de commande. Cependant le `network-manager` coupe et relance la connexion Wi-Fi si elle était déjà opérationnelle : ce n'est pas ce que nous voulons.

Pourquoi dès lors ne pas vérifier si le dispositif n'est pas relié au point d'accès avant de tenter une connexion ? Parce que le `network-manager` garde en mémoire pendant un certain temps la connexion précédente comme si elle était active même si le point d'accès glisse hors de portée. Ce temps de latence peut atteindre 6 minutes dans le pire des cas [2] et est indépendant de la périodicité du *scan*. Nous ne pouvons donc pas nous fier aux informations de la connexion actuelle pour lancer un *scan* et, *in fine*, pour forcer la connexion en ligne de commande. Ces informations non mises à jour posent donc problème pour le passage d'un point d'accès à un autre.

Une autre possibilité est d'utiliser une méthode plus "*legacy*" pour configurer les interfaces. Elle se base sur les outils `iwconfig`

1. Le balayage actif s'oppose au balayage passif dans lequel la STA écoute les *beacons* diffusés à intervalle régulier par les points d'accès [21].

2. 6 minutes est la valeur la plus élevée que nous avons obtenu à plusieurs reprises lors de quelques tests.

et `iwlist` provenant du *package* `wireless-tools` [35]. La commande `iwlist` permet d'obtenir la liste des réseaux disponibles alors que `iwconfig` autorise une configuration plus poussée de la carte Wi-Fi que ne le fait le `network-manager`. Ce dernier propose quant à lui une interface de plus haut niveau.

Un problème se pose cependant. Les scripts[3] `mptcp_up` et `mptcp_down` lancés respectivement à chaque fois qu'une interface est activée ou désactivée se basent sur les informations de l'interface livrées par le network-manager.

Or, pour utiliser `iwconfig` et `iwlist`, il est nécessaire de stopper le `network-manager` car il bloque l'utilisation d'un autre programme sur la même carte réseau.

Nous avons donc été obligés de réécrire ces scripts afin de récupérer les informations d'une autre manière. Ainsi, l'adresse IP de l'interface a, par exemple, été obtenue en "parsant" l'output de la commande `ifconfig` et le *gateway* de l'interface dans le fichier de *lease* de la commande `dhclient`[4]. Ces scripts modifiés sont disponibles en annexe au point 8.2.

Nous avons obtenu de meilleurs résultats avec les outils du *package* `wireless-tools` qu'avec l'utilisation du `network-manager`. En calculant le temps mis depuis la génération du hotspot jusqu'à sa détection, il nous a fallu 3 secondes lors des 10 essais, soit le temps pris par la commande *iwlist* pour s'exécuter.

6.3 Problématique du NAT

6.3.1 Problème

Nous avons utilisé une clé Wi-Fi externe en guise d'interface Wi-Fi mais nous aurions également pu utiliser un smartphone en *tethering* qui partagerait sa connexion Wi-Fi. Cette architecture est également une solution possible au problème de périodicité du *scan* débattu au point 6.2.2.

Cependant, en utilisant un smartphone, le changement d'adresse IP lors d'un *handover* Wi-Fi est masqué par la table `NAT` du téléphone.

3. Ces scripts permettent la mise en place automatique des tables de routages comme expliqué au point 2.6.1 du chapitre 2

4. dhclient est un programme qui permet de contacter un serveur DHCP afin de d'obtenir une adresse IP sur le réseau. Elle récupère également des informations relative au réseau telles que la route par défaut (gateway) et le serveur DNS à contacter notamment.

La machine virtuelle Debian n'observe donc pas le changement d'adresse IP et doit attendre l'expiration du `Retransmission Time Out` (RTO). Cette opération implique un temps de latence au niveau du *handover* MPTCP car le *subflow* n'est pas directement recréé, comme expliqué au point 2.6.3 du chapitre 2.

Nous n'observons pas ce problème pour l'interface 4G du dispositif expérimental puisqu'elle repose sur le *handover* cellulaire et conserve donc son adresse IP lors de la reprise du transfert.

6.3.2 Solutions possibles

L'utilisation du NAT pose un sérieux problème au niveau de la continuité de la connexion MPTCP lors d'un *handover* Wi-Fi.

Pour optimiser le temps de récupération, il est alors nécessaire de faire 2 vérifications :

— vérifier que la relation avec l'ancienne adresse IP dans la table NAT est bien supprimée et que le smartphone renvoie bien un `RST` dans le cas où un paquet lui est envoyé ;
— vérifier que le serveur contacté par l'appareil mobile renvoie bien un `RST` quand il reçoit un paquet pour une connexion non existante.

Pour réaliser cette première vérification il est nécessaire de "rooter" le smartphone. Ce point n'a pas pu être expérimenté mais a été validé de manière théorique par un chercheur de l'UCL qui travaille sur MPTCP.

6.4 Authentification aux portails

6.4.1 Problème

Lors de nos mesures expérimentales sur le *streaming* audio dans les trains du point 5.3, nous avons été contraints d'utiliser un script pour récupérer la page du portail via l'interface Wi-Fi. Nous discutons, plus en détail de ce problème dans cette partie.

TZEVELECOS avait déjà soulevé le problème lors de son mémoire [36]. Il avait constaté que l'utilisation de Multipath TCP empêchait la récupération de la page d'authentification sur les bornes `TELENETHOTSPOT`. Nous avons également utilisé ces points d'accès lors de nos mesures de *streaming* audio du chapitre 5.

Selon la RFC8041 [7], un smartphone avec une interface Wi-Fi et une cellulaire va majoritairement avoir tendance à choisir l'interface Wi-Fi pour la route par défaut dans la table de routage générale.

Ce document indique que cette approche n'est pas optimale pour MPTCP dans le cas où l'interface Wi-Fi se trouve en connexion limitée, e.g. derrière un portail d'accès. Effectivement, en créant le *subflow* initial, le *kernel* va choisir l'interface Wi-Fi et faillir dans sa tâche. Un comportement plus adapté serait de tester plusieurs fois sur cette interface et en cas d'un échec, essayer sur la deuxième également.

Contrairement à cet article, lors de nos essais, l'interface choisie par défaut est toujours celle de l'interface 4G (usb0) au lieu de l'interface Wi-Fi (wlan0). Et ce, peu importe l'ordre dans lequel les interfaces sont activées.

Ce choix pose évidemment problème car il nous empêche d'accéder au portail d'accès afin de compléter l'authentification.

6.4.2 Solutions envisagées

Pour résoudre le problème d'accès à la page d'authentification des portails, TZEVELECOS proposait dans son mémoire [36] de :

1. désactiver l'interface cellulaire ;
2. se connecter à la borne Wi-Fi ;
3. redémarrer le navigateur Internet ;
4. entrer les informations d'authentification sur la page du portail ;
5. réactiver l'interface cellulaire.

Cette solution, bien que fonctionnelle, ne permet pas d'obtenir une continuité lors du transfert du *streaming*, vidéo dans le cas de ce mémoire. En effet, bien que MPTCP soit utilisé pleinement dès lors que cette manipulation est effectuée, il induit une coupure des deux interfaces pendant un certain laps de temps.

Nous proposons dans notre mémoire plusieurs solutions qui tiennent compte du caractère continu que l'on désire mettre en place au niveau du *streaming* audio sans devoir couper l'une ou l'autre interface.

Ainsi, il existe plusieurs possibilités [5] pour atteindre la page d'authentification des portails :

1. changer la route par défaut de la table de routage générale vers l'interface Wi-Fi ;

5. La liste des solutions possibles est non exhaustive

2. utiliser `iptables` pour diriger les paquets d'un *user/process id*/port de connexion vers le gateway de l'interface Wi-Fi [20]. Cette deuxième solution ne touche pas à la route par défaut de la table de routage générale ;
3. choisir l'IP Source des paquets et indiquer l'adresse IP de l'interface Wi-Fi. Cette solution requiert d'avoir un accès au code source de l'application ;
4. utiliser un logiciel qui redirige les paquets d'un programme sur l'interface Wi-Fi. Les outils `force_bind`[6] et `ForceBindIP`[7] sont disponibles respectivement sous Linux et Windows.

Étant donné que nous avions besoin de remplir les informations d'authentification le plus rapidement possible, nous avons créé un script qui automatise ce processus. Il est écrit en Python et spécifie l'adresse IP de l'interface Wi-Fi comme IP Source pour récupérer la page du portail. Il renvoie alors le formulaire complété pour finaliser l'authentification.

6. http ://freecode.com/projects/force_bind/
7. https ://r1ch.net/projects/forcebindip

Chapitre 7

Conclusion

Ce mémoire, tiré d'une réflexion théorique, s'est avéré plus difficile à réaliser dans la pratique. Les mesures obtenues ont permis de mettre en évidence des éléments clés dans la mise en place et l'utilisation du Multipath TCP (MPTCP) dans les transports sur voie ferrée.

Après un état de l'art présentant le protocole, le chapitre 3 a proposé un dispositif expérimental permettant l'utilisation de MPTCP dans une machine virtuelle Debian disposant de 2 interfaces réseau. Une interface 4G est obtenue en *tethering* à l'aide d'un smartphone alors qu'une interface Wi-Fi est soutenue par une clé Wi-Fi externe. Les choix d'une telle architecture ont également été exposés.

Ensuite, en terme d'environnement, le chapitre 4 nous a démontré que l'utilisation de MPTCP est plus propice dans les transports disposant de points d'accès prévus pour la clientèle aux arrêts. Les trains en Wallonie, ainsi que les métros bruxellois, disposent de cette infrastructure. Elle procure une puissance de signal suffisamment forte pour permettre la connexion, contrairement aux hostpots publics générés par les particuliers.

Malgré la diversité des environnements analysés, il serait particulièrement envisageable dans le futur de faire des mesures dans des cadres qui baignent le dispositif expérimental d'ondes Wi-Fi pendant toute la durée des trajets. En effet, actuellement la plupart des trains à grande vitesse (TGV) proposent cette infrastructure. XIAO et al. ont réalisé des mesures avec TCP sur de la 4G [39] dans les TGV mais aucun test n'a encore été réalisé avec MPTCP dans ce contexte. De plus, la SNCB a également réalisé des tests d'introduction de bornes Wi-Fi dans les trains [33] et en proposera probablement à l'avenir. Ces analyses pourraient mettre en évidence en quelle mesure MPTCP, combiné à la 4G, pourrait désengorger le réseau Wi-Fi du train lors des trajets à forte affluence.

Au cours du chapitre 5, nous avons réalisé des mesures lors de *streaming* audio dans les métros et les trains. Suite à celles-ci, il nous est possible de dire que MPTCP est plus utile dans les métros

que dans les trains en observant le temps d'utilisation des deux interfaces par rapport à l'utilisation de la 4G seule lors des trajets.

Enfin, il ressort du chapitre 6 sur la critique du dispositif expérimental, que le gestionnaire de réseau de la distribution Debian semble ne pas avoir été conçu dans un esprit de mobilité. En effet, celui-ci a des difficultés à se rendre compte que le point d'accès n'est plus à portée et n'essaie donc pas de se reconnecter. Ce problème, dû a une périodicité de *scan* des réseaux insuffisante, entrave le fonctionnement des scénarios mobiles. Nous avons également décelé un problème lorsqu'un *handover* MPTCP est réalisé derrière un NAT (donc caché au dispositif expérimental) et envisagé des solutions. Enfin, l'authentification aux portails d'accès est également discutée dans ce chapitre.

Nous concluons ce mémoire en relevant que notre dispositif expérimental n'était pas le mieux adapté pour réaliser des mesures dans un contexte de mobilité. En effet, bien que cette architecture soit tout à fait envisageable dans les trains, elle l'est bien moins en tram ou en métro, où son utilisation est moins aisée. Cependant ce dispositif nous a permis de cerner les limites du gestionnaire de réseau et de proposer quelques solutions.

Dans le futur, il serait intéressant de réaliser à nouveau des mesures dans ces différents environnements, soit en modifiant le dispositif pour qu'il tienne compte des remarques et solutions possibles envisagées, soit en utilisant un smartphone directement embarqué avec MPTCP.

Chapitre 8

Annexes

8.1 Script de connexion automatique aux portails

Script python réalisé dans le but de résoudre le problème exposé au point 6.4.1 du chapitre 6. Il est basé sur un programme écrit par PUYBAREAU en 2008 et disponible à cette adresse [27].

```
#!/usr/bin/env python

################################################################################
# OpenWifiAutoConnect - Stephane PUYBAREAU (puyb <at> puyb <dot> net) - 2008   #
# Fully automated authentication to capture portal based wifi networks.        #
# Config file: ~/.OpenWifiAutoConnect - Format: ini                            #
# Section are network SSID (names)                                             #
# key / value pair define user submited information (based on the html form)   #
#                                                                              #
# This software is provided under the terms of the GPL v3 licence.             #
# See http://www.gnu.org/licenses/gpl.html for more information                #
#                                                                              #
# This software use python dbus bindings, pynotify and BeautifulSoup modules   #
################################################################################

# Script modified by Laurent Miny to be used in multipath environnement for a master thesis
# Original script on http://puyb.net/2009/12/open-wifi-auto-connect/
#
# Modifications made :
# - Added a loop to try multiple URL hoping to be redirected to the portal page
# - Modified the use of urllib2 to specify a source IP (for MPTCP)
# - Made the ConfigParser aware of the case
# - Added another way to know if we are connected on the Internet (has_internet function)
# - Made multiples optimisation to be able to use it in the environment of the master thesis
#
# Requirements : beautifulsoup, netifaces, dbus, pynotify
#
# License : GPL v3 (http://www.gnu.org/licenses/gpl.html)

import subprocess
import re
import time
import logging
import functools
import httplib
import urllib2, urllib
from BeautifulSoup import BeautifulSoup
import re
import os
import ConfigParser
import sys
import gobject
import dbus
import dbus.mainloop.glib
import pynotify
import socket

##################
# CONFIGURATION #
##################
SSID = 'PROXIMUS_FON'
INTERFACE = 'wlan0'
# The urls the script will try to open hoping to be redirected to the portal
URLs = ['http://www.wifi.brussels', 'http://detectportal.firefox.com', 'http://www.apple.com',
        ↪ 'http://www.facebook.com', 'http://www.google.com', 'http://www.microsoft.com',
        ↪ 'http://www.amazon.com', 'http://www.evernote.com', 'http://www.spotify.com',
        ↪ 'http://www.wifi.brussels', 'http://www.youtube.com', 'http://www.voomotion.be',
        ↪ 'http://www.proximus.com']
CONFIGFILE = '../connect_portal.ini'
```

```
#############
# FUNCTIONS #
#############

def get_config(path):
    config = ConfigParser.ConfigParser()
    config.optionxform=str
    config.read(os.path.expanduser(path))
    return config

def get_ip(interface) :
    import netifaces as ni
    ni.ifaddresses(interface)
    ip = ni.ifaddresses(interface)[ni.AF_INET][0]['addr']
    return ip

# To force sending packet on a particular interface by setting the IP Source
class BoundHTTPHandler(urllib2.HTTPHandler):
    # Solution from https://stackoverflow.com/a/14669175
    def __init__(self, source_address=None, debuglevel=0):
        urllib2.HTTPHandler.__init__(self, debuglevel)
        self.http_class = functools.partial(httplib.HTTPConnection, source_address=source_address)

    def http_open(self, req):
        return self.do_open(self.http_class, req)

def has_internet(interface) :
    handler = BoundHTTPHandler(source_address=(get_ip(interface), 0))
    opener = urllib2.build_opener(handler, urllib2.HTTPCookieProcessor())
    opener.addheaders = [('User-Agent', 'Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101
        ↪ Firefox/52.0')]
    urllib2.install_opener(opener)

    f = opener.open('http://www.google.com')
    data = f.read()
    f.close()
    soup = BeautifulSoup(data)
    if soup.title.string == 'Google' :
        print "Connected !"
        return True
    else :
        return False

def properties_changed_signal_handler(props):
    config = get_config(CONFIGFILE)
    if not props.has_key('ActiveConnections'):
        return
    for device_path in props['ActiveConnections']:
        device = bus.get_object('org.freedesktop.NetworkManager', device_path)
        device_props = device.GetAll("org.freedesktop.NetworkManager.Connection.Active",
            ↪ dbus_interface="org.freedesktop.DBus.Properties")
        if device_props['Default']:
            return
        ap_path = device_props['SpecificObject']
        if ap_path.startswith('/org/freedesktop/NetworkManager/AccessPoint/'):
            ap = bus.get_object('org.freedesktop.NetworkManager', ap_path)
            ssid = ap.Get("org.freedesktop.NetworkManager.AccessPoint", "Ssid",
                ↪ dbus_interface="org.freedesktop.DBus.Properties")
            ssid = ''.join([chr(c) for c in ssid])
            if ssid not in config.sections():
                return
            device.connect_to_signal("PropertiesChanged",
                ↪ device_properties_changed_signal_handler(ssid),
                ↪ dbus_interface="org.freedesktop.NetworkManager.Connection.Active")

def device_properties_changed_signal_handler(ssid):
    def handler(props):
        #print "Device_prop_changed"
        #print ssid
        if not props.has_key('State'):
            return
        if props['State'] != 2:
            return

        status = login(SSID, INTERFACE)
        if status == True:
            txt = "Successfully logged on " + ssid
        else:
            txt = "Failed to log on " + ssid
        n = pynotify.Notification("Open Wifi Auto Connect", txt, "dialog-warning")
        n.set_urgency(pynotify.URGENCY_NORMAL)
        n.set_timeout(10)
        #n.add_action("clicked","Button text", callback_function, None)
        n.show()
    return handler

def login(ssid, interface) :
    config = get_config(CONFIGFILE)
    credentials = dict(config.items(ssid))

    if has_internet(interface) :
        print "Has already internet access"
        return True

    print "Login"

    handler = BoundHTTPHandler(source_address=(get_ip(interface), 0))
```

```
    opener = urllib2.build_opener(handler, urllib2.HTTPCookieProcessor())
    opener.addheaders = [('User-Agent', 'Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101
        ↪ Firefox/52.0')]
    urllib2.install_opener(opener)

    # Try to get portal page
    for url in URLs :
        # try to open the portal page
        print "try url: " + url
        f = opener.open(url)
        data = f.read()
        redirect_url = f.geturl().split('?')[0]
        f.close()
        if 'portal' in redirect_url and 'detectportal' not in redirect_url :
            print "Portal found! " + redirect_url
            break

    # parse the portal page
    soup = BeautifulSoup(data)
    form = None
    for link in soup.findAll('form'):
        if len(link['action']) > 8 :
            form = link
            break

    if not form:
        return # There's no form on this page

    # creating the post credentials
    login_post = {}
    for input in form.findAll('input'):
        if input.has_key('name'):
            default = ''
            if input.has_key('type') and input['type'] == 'checkbox':
                default = 'on'
            login_post[input['name']] = input.has_key('value') and input['value'] or default

    login_post.update(credentials)

    url = form['action']

    # GET or POST ?
    postBody = None
    if form.has_key('method') and form['method'].lower() == 'post':
        postBody = urllib.urlencode(login_post)
    else:
        url += '?' + urllib.urlencode(login_post).replace(' ', '+')

    # submit the form
    f = opener.open(url, postBody)
    data = f.read()
    f.close()

    return has_internet(interface)



########
# MAIN #
########

if __name__ == '__main__':

    pynotify.init("Open Wifi Auto Connect")

    dbus.mainloop.glib.DBusGMainLoop(set_as_default=True)

    bus = dbus.SystemBus()
    nm = bus.get_object("org.freedesktop.NetworkManager", "/org/freedesktop/NetworkManager")
    nm.connect_to_signal("PropertiesChanged", properties_changed_signal_handler,
        ↪ dbus_interface="org.freedesktop.NetworkManager")

    loop = gobject.MainLoop()
    loop.run()
```

8.2 Scripts de configuration du routage sans network-manager

Afin de configurer automatiquement le routage sans l'utilisation des informations du network-manager, il est nécessaire de modifier les scripts `mptcp_up` et `mptcp_down` disponibles sur le site internet de l'implémentation [32].

8.2.1 mptcp_up.sh

```
#!/bin/bash
# A script for setting up routing tables for MPTCP (only for IPv4)

# Param : $1 = Interface (e.g. wlan0, usb0)

_V=0

DEVICE_IFACE=$1

while getopts "v" OPTION
do
  case $OPTION in
    v) _V=1; DEVICE_IFACE=$2
       ;;
  esac
done


if [ -z $DEVICE_IFACE ] ; then
    echo 'No arguments for interface'
    exit 0
fi

set -e

# Collect information without network-manager
IP4_ADDRESS_0=$(sudo ip addr show $DEVICE_IFACE | grep 'inet ' | awk '{print($2)}')
DHCP4_IP_ADDRESS=$(sudo ifconfig $DEVICE_IFACE | grep 'inet ' | awk '{print($2)}')
NETMASK=$(sudo ifconfig $DEVICE_IFACE | grep 'inet ' | awk '{print($4)}')
SUBNET=`echo $IP4_ADDRESS_0 | cut -d \.  -f 1 | cut -d / -f 2`

# From https://stackoverflow.com/a/15429733/5955402
IFS=. read -r i1 i2 i3 i4 <<< $DHCP4_IP_ADDRESS
IFS=. read -r m1 m2 m3 m4 <<< $NETMASK
DHCP4_NETWORK_NUMBER=$(printf "%d.%d.%d.%d\n" "$((i1 & m1))" "$((i2 & m2))" "$((i3 & m3))" "$((i4 &
    ↪ m4))")

# To get the Gateway
# How to : get the last lease for the interface DEVICE_IFACE (it can have old ones) and parse the
    ↪ option routers to get the gateway
# line = line of last lease for interface DEVICE_IFACE
line=$(grep -n $DEVICE_IFACE /var/lib/dhcp/dhclient.leases | tail -n 1 | awk '{print($1)}' | sed
    ↪ 's/:$//')
if [ -z $line ] ; then
    echo 'No Gateway found for $DEVICE_IFACE'
    exit 0
fi
let "end_line=$line+7"
DHCP4_ROUTERS=$(head -n $end_line /var/lib/dhcp/dhclient.leases | tail -n 8 | grep routers | awk
    ↪ '{print($3)}' | sed 's/;$//')

if [ $_V -eq 1 ]; then
    echo "IP4_ADDRESS_0=$IP4_ADDRESS_0"
    echo "DHCP4_ROUTERS=$DHCP4_ROUTERS"
    echo "DHCP4_NETWORK_NUMBER=$DHCP4_NETWORK_NUMBER"
    echo "DHCP4_IP_ADDRESS=$DHCP4_IP_ADDRESS"
    echo "NETMASK=$NETMASK"
    echo "SUBNET=$SUBNET"
fi

# Check if all variables contain information
if [ -z $DEVICE_IFACE ] || [ -z $IP4_ADDRESS_0 ] || [ -z $DHCP4_IP_ADDRESS ] || [ -z $DHCP4_ROUTERS
    ↪ ] || [ -z $NETMASK ] || [ -z $SUBNET ]; then
    echo "Some information is missing"
    exit 0
fi

# FIRST, make a table-alias
if [ `grep $DEVICE_IFACE /etc/iproute2/rt_tables | wc -l` -eq 0 ]; then
    NUM=`cat /etc/iproute2/rt_tables | wc -l`
    echo "$NUM $DEVICE_IFACE" >> /etc/iproute2/rt_tables
fi

# Add routes (only IPv4)
ip route add table $DEVICE_IFACE to $DHCP4_NETWORK_NUMBER/$SUBNET dev $DEVICE_IFACE scope link
```

```
ip route add table $DEVICE_IFACE default via $DHCP4_ROUTERS dev $DEVICE_IFACE
ip rule add from $DHCP4_IP_ADDRESS table $DEVICE_IFACE
```

8.2.2 mptcp_down.sh

```
#!/bin/bash
# A script for setting up routing tables for MPTCP

# Param : $1 = Interface (e.g. wlan0, usb0)

set -e

DEVICE_IFACE=$1

if [ -z $DEVICE_IFACE ]; then
        echo "Some information is missing"
    exit 0
fi

ip rule del table $DEVICE_IFACE
ip route flush table $DEVICE_IFACE
```

Bibliographie

[1] TMA 2016. *Multipath TCP - Olivier Bonaventure (Université catholique de Louvain)*. Youtube. 2016. URL : https://youtu.be/PkvLq_kCv4o?t=56m36s.

[2] Saba AHSAN et al. *Multipath RTP (MPRTP)*. Internet-Draft draft-ietf-avtcore-mprtp-03. Work in Progress. Internet Engineering Task Force, juil. 2016. 41 p. URL : https://datatracker.ietf.org/doc/html/draft-ietf-avtcore-mprtp-03.

[3] Sébastien BARRÉ, Christoph PAASCH et Olivier BONAVENTURE. « MultiPath TCP : From Theory to Practice ». In : *NETWORKING 2011 : 10th International IFIP TC 6 Networking Conference, Valencia, Spain, May 9-13, 2011, Proceedings, Part I*. Sous la dir. de Jordi DOMINGO-PASCUAL et al. Berlin, Heidelberg : Springer Berlin Heidelberg, 2011, p. 444–457. ISBN : 978-3-642-20757-0. DOI : 10.1007/978-3-642-20757-0_35. URL : http://dx.doi.org/10.1007/978-3-642-20757-0_35.

[4] Dinamene de Lima Correia Almeida BARREIA. « Multipath TCP Protocols ». Mém.de mast. Técnico Lisboa, 2014.

[5] Olivier BONAVENTURE. *Multipath RTP*. 2013. URL : https://perso.uclouvain.be/olivier.bonaventure/blog/html/2013/11/30/multipath_rtp.html.

[6] Olivier BONAVENTURE. *Why is the Multipath TCP scheduler so important ?* Mar. 2014. URL : http://blog.multipath-tcp.org/blog/html/2014/03/30/why_is_the_multipath_tcp_scheduler_so_important.html.

[7] Olivier BONAVENTURE, Christoph PAASCH et Gregory DETAL. *Use Cases and Operational Experience with Multipath TCP*. RFC 8041. Jan. 2017. DOI : 10.17487/RFC8041. URL : https://rfc-editor.org/rfc/rfc8041.txt.

[8] CIRB. *Wifi.brussels et la STIB*. Avr. 2016. URL : http://cirb.brussels/fr/quoi-de-neuf/actualites/wifi-brussels-et-la-stib.

[9] Gerald COMBS. *TShark*. URL : https://www.wireshark.org/docs/man-pages/tshark.html.

[10] Quentin DE CONINCK et Matthieu BAERTS. « Multipath TCP with real Smartphone applications ». Prom. : Bonaventure, Olivier. Mém.de mast. Ecole polytechnique de Louvain, Université catholique de Louvain, 2015.

[11] Jean-Michel DRICOT. *Cours de Mobile & Wireless Networks*. 2016.

[12] Alan FORD et al. *TCP Extensions for Multipath Operation with Multiple Addresses*. RFC 6824. Oct. 2015. DOI : 10.17487/rfc6824. URL : https://rfc-editor.org/rfc/rfc6824.txt.

[13] GNOME FOUNDATION. *Network-manager*. 2004-2017. URL : https://wiki.gnome.org/Projects/NetworkManager.

[14] FREEDESKTOP.ORG. *Network-manager source code*. 2017. URL : https://cgit.freedesktop.org/NetworkManager/NetworkManager/tree/src/devices/wifi/nm-device-wifi.c?id=3bd5a83eff69a261cf34f17424b3152ea11130d0.

[15] Stephen HEMMINGER et Alexey KUZNETSOV. *iproute2*. 2017. URL : https://wiki.linuxfoundation.org/networking/iproute2.

[16] IBPT. *Cartes de couverture : réseaux mobiles*. 2017. URL : http://www.bipt.be/fr/consommateurs/telephone/qualite-de-service/cartes-de-couverture-reseaux-mobiles.

[17] INFRABEL. *Document de Référence du Réseau*. Jan. 2017. URL : https://www.infrabel.be/sites/default/files/documents/drr_2017_20170127.pdf.

[18] IRISNET. *wifi.brussels*. URL : http://wifi.brussels/fr/.

[19] Janardhan IYENGAR et al. *Architectural Guidelines for Multipath TCP Development*. RFC 6182. Oct. 2015. DOI : 10.17487/rfc6182. URL : https://rfc-editor.org/rfc/rfc6182.txt.

[20] JIKAN. *Forcer une application à utiliser une interface réseau donnée sur gnulinux ou comment utiliser deux connections Internet en même temps*. Mar. 2013. URL : https://jikan.fr/forcer-une-application-a-utiliser-une-interface-reseau-donnee-sur-gnulinux-ou-comment-utiliser-deux-connections-internet-en-meme-temps/.

[21] J. KUROSE et K. ROSS. *Computer Networking : A Top-Down Approach, Global Edition*. Pearson Education Limited, 2017. ISBN : 9781292153605.

[22] LUXUL. *Wireless AP Placement*. 2017. URL : https://luxul.com/ap-placement.

[23] Christoph PAASCH. « Improving Multipath TCP ». Thèse de doct. UCLouvain / ICTEAM / EPL, 2014.

[24] Christoph PAASCH et Olivier BONAVENTURE. « Multipath TCP ». In : *Commun. ACM* 57.4 (avr. 2014), p. 51–57. ISSN : 0001-0782. DOI : 10.1145/2578901. URL : http://doi.acm.org/10.1145/2578901.

[25] Christoph PAASCH et al. « Experimental evaluation of multipath TCP schedulers ». In : *Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop*. ACM. 2014, p. 27–32.

[26] Christoph PAASCH et al. « Exploring Mobile/WiFi Handover with Multipath TCP ». In : *ACM SIGCOMM workshop on Cellular Networks (Cellnet'12)*. 2012.

[27] Stephane PUYBAREAU. *Open Wifi Auto Connect*. Déc. 2009. URL : http://puyb.net/2009/12/open-wifi-auto-connect/.

[28] Costin RAICIU et al. « How Hard Can It Be ? Designing and Implementing a Deployable Multipath TCP ». In : *USENIX Symposium of Networked Systems Design and Implementation (NSDI'12), San Jose (CA)*. 2012.

[29] Costin RAICIU et al. « Opportunistic Mobility with Multipath TCP ». In : *Proceedings of the Sixth International Workshop on MobiArch*. MobiArch '11. Bethesda, Maryland, USA : ACM, 2011, p. 7–12. ISBN : 978-1-4503-0740-6. DOI : 10.1145/1999916.1999919. URL : http://doi.acm.org/10.1145/1999916.1999919.

[30] Rami ROSEN. *Linux Kernel Networking course*. 2007. URL : http://www.haifux.org/lectures/172/netLec.pdf.

[31] La RÉDACTION. *Le nombre de smartphones vendus dans le monde*. 2016. URL : http://www.journaldunet.com/ebusiness/internet-mobile/1009563-le-nombre-de-smartphones-vendus-dans-le-monde/.

[32] C. Paasch S. BARRE et al. *Multipath TCP in the Linux Kernel*. URL : https://www.multipath-tcp.org.

[33] SNCB. *La clientèle de la SNCB va tester le wifi à bord de certains trains*. Fév. 2016. URL : http://www.belgianrail.be/fr/corporate/Presse/Presse-releases/25_02_2016.aspx.

[34] T0MM13B. *Where can I find settings for wifi internal scan period*. Jan. 2013. URL : https://android.stackexchange.com/questions/37621/where-can-i-find-settings-for-wifi-internal-scan-period.

[35] Jean TOURRILHES. *wireless-tools*. 2006. URL : https://packages.debian.org/fr/jessie/wireless-tools.

[36] Takis TZEVELECOS. « Dynamic Adaptive Streaming over HTTP (DASH) with MPTCP ». Mém.de mast. Université de Namur, 2013.

[37] VIDEOLAN. *VLC Media Player*. URL : https://www.videolan.org/vlc/index.fr.html.

[38] Damon WISCHIK et al. « Design, Implementation and Evaluation of Congestion Control for Multipath TCP ». In : *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*. NSDI'11. Boston, MA : USENIX Association, 2011, p. 99–112. URL : http://dl.acm.org/citation.cfm?id=1972457.1972468.

[39] Qingyang XIAO et al. « TCP performance over mobile networks in high-speed mobility scenarios ». In : *2014 IEEE 22nd International Conference on Network Protocols*. IEEE. 2014, p. 281–286.

[40] XIPH.ORG. *Icecast*. 2004-2017. URL : http://icecast.org/.

[41] XIPH.ORG. *Ices*. 2004-2017. URL : http://icecast.org/ices/.

[42] Lesmana ZIMMER. *How to change the refresh rate at which the network manager updates the wireless*. 2011. URL : https://askubuntu.com/questions/53498/how-to-change-the-refresh-rate-at-which-the-network-manager-updates-the-wireless.